Aula 01

Apresentação da Disciplina;

Introdução aos Sistemas Operacionais;

Histórico.

Definição

"... um conjunto de rotinas executadas pelo processador, de forma semelhante aos programas dos usuários."

"... O SO tem por objetivo funcionar como uma interface entre o usuário e o computador, tornando sua utilização mais simples, rápida e segura".

Francis Machado e Luiz Paulo Maia

Definição

"... é um programa que atua como intermediário entre o usuário e o hardware de um computador."

"... deve propiciar um ambiente no qual o usuário possa executar programas de forma conveniente e eficiente".

Silberschatz, Galvin e Gagne

Características de um SO

Transparência

- Simplifica as Atividades para o usuário

Gerência

- Compartilha e Otimiza os recursos computacionais

Encapsulamento

- Esconde os detalhes da implementação;
- “Padroniza" a interface, muitas vezes mesmo entre dispositivos diferentes.

Evolução dos SOs

Nos primeiros dispositivos computacionais:

- Operação complexa
  - Já ouviram falar do cargo de “Operador de Computador” ?
- Programação ainda mais complexa
  - Exigia grande conhecimento do *hardware* , inclusive da "linguagem de máquina”

Atualmente, SOs tendem à linguagem natural

Filme de 1986 (a 35 anos atrás)

Prof. Marco Câmara

Em 2021 …

Exemplos SO (computadores pessoais)

Windows (Windows 11) 69,52%

MacOS (Versão atual: Ventura; breve “Sonoma") 20,42%

”Não sei" 3,69%

Linux (MX Linux, Debian, Manjaro, Ubuntu) 3,07%

ChromeOS 4,13% -> **Total > 100% ?**

Unix (FreeBSD, SCOUnix, HP-UX, SunOS) 0% ?

https://gs.statcounter.com/os-market-share/mobile/worldwide

Exemplos SO (*smartphones*)

Android (70,9%)

iOS (28,36%)

Samsung (0,38%)

KaiOS (0,15%)

Não sei (0,19%)

Windows (0,02%)

https://gs.statcounter.com/os-market-share/mobile/worldwide

Exemplos SO (outros ambientes)

Workstations de aplicação específica

Servidores

Eletrodomésticos (SO embarcados)

Computadores de grande porte (*mainframes*)

Exemplos SO

Quais as principais diferenças?

- Quem conhece mais de um sistema operacional?
- Apresentar diferenças.

Aula 02

Tipos de SO;

Histórico dos SOs;

Máquina de Níveis;

Conceitos de *Hardware.*

Tipos de Sistemas Operacionais

Monoprogramáveis / Monotarefa

Multiprogramáveis / Multitarefa

Multiprocessados

Tipos de Sistemas Operacionais

- Sistemas Monoprogramáveis / Monotarefa
  - Todos recursos do sistema dedicados a uma tarefa
  - Execução de programas seqüencialmente
  - Usado nos primeiros computadores de grande porte
  - Usado nos primeiros computadores de pequeno porte
    - Sistemas Monousuário

Tipos de Sistemas Operacionais

- Sistemas Monoprogramáveis / Monotarefa
  - Sub-utilização dos recursos do sistema
    - Processador X Operações de E/S
    - Memória
    - Dispositivos de E/S
  - Implementação simples
    - Sem recursos de proteção

Tipos de Sistemas Operacionais

- Sistemas Multiprogramáveis / Multitarefa
  - Recursos do sistema compartilhados por diversas tarefas
  - Execução de programas concorrentemente
    - Aumento da produtividade
    - Redução de custos
    - Suporte a Sistemas Multiusuário
  - Compartilhamento na utilização dos recursos do sistema
    - Processador X Operações de E/S
    - Memória
    - Dispositivos de E/S

Tipos de Sistemas Operacionais

- Sistemas Multiprogramáveis / Multitarefa
  - Implementação complexa
    - Gerenciamento dos acessos concorrentes aos recursos
    - Recursos de proteção
  - Sistemas Monousuário X Multiusuário
    - Mainframes
    - Computadores Pessoais e Estações de Trabalho

Tipos de Sistemas Operacionais

- Sistemas Multiprogramáveis / Multitarefa
  - Sistemas de Tempo Real
    - Semelhante aos Sistemas de Tempo Compartilhado
    - Limites rígidos para tempo de resposta
    - Sem fatia de tempo
    - Níveis de prioridade
    - Aplicações multimídia interativas

Tipos de Sistemas Operacionais

- Multitarefa
  - Colaborativa
    - Windows 95, 98
    - Não existe fatia de tempo
  - Preemptiva
    - Windows 10 e Server, Linux, Mac OS, ...
    - Existe fatia de tempo
    - Preempção

Tipos de Sistemas Operacionais

- Sistemas Multiprocessados
  - Sistemas com mais de uma CPU interligada
    - Execução simultânea de programas
    - Supre dificuldade no desenvolvimento de processadores mais rápidos
    - Ideal para sistemas que necessitam uso intensivo de CPU
      - Processamento científico
    - ... ou para uso diferenciado de CPU
      - Baixo consumo, ou processamento avançado? (celulares com múltiplas câmeras ?)

Tipos de Sistemas Operacionais

- Sistemas Multiprocessados
  - Características
    - Multiprogramação
      - Aplicada a cada processador
    - Escalabilidade
      - Aumento da capacidade computacional
    - Reconfiguração
      - Tolerância à falha em algum processador
    - Balanceamento
      - Distribuição de carga de processamento

Tipos de Sistemas Operacionais

- Sistemas Multiprocessados
  - Classificação
    - Em função:
      - da forma de comunicação entre CPUs
      - do grau de compartilhamento da memória e E/S
    - Sistemas Fortemente Acoplados
      - Processadores com múltiplos núcleos, por exemplo.
    - Sistemas Fracamente Acoplados
      - *Server Cluster*, por exemplo.

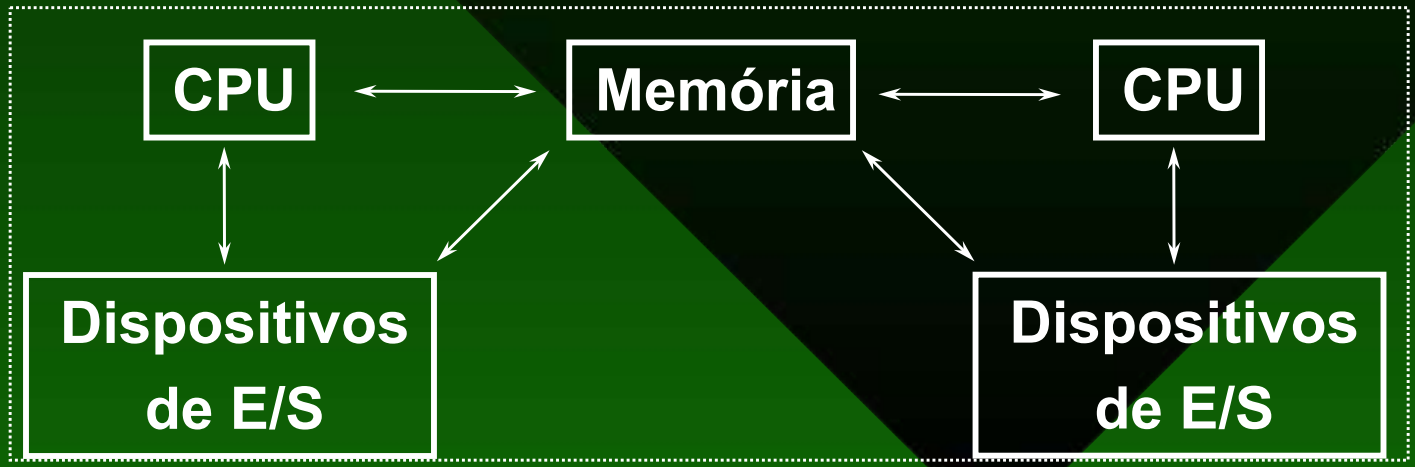
Tipos de Sistemas Operacionais

Sistemas Multiprocessados

Sistemas Fortemente Acoplados

Processadores compartilham um única memória

Espaço de Endereçamento Único
Único SO



Tipos de Sistemas Operacionais

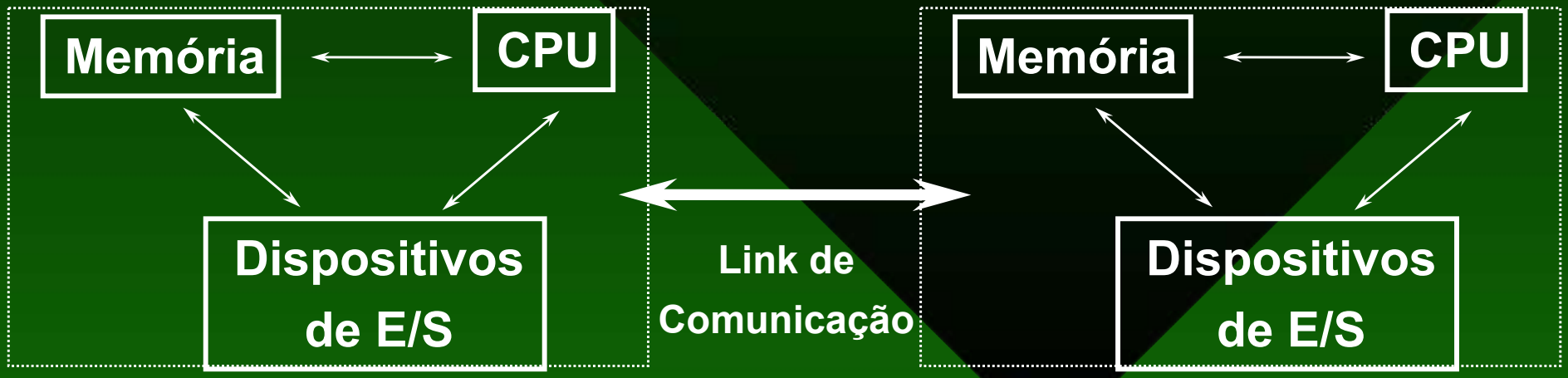
Sistemas Multiprocessados

Sistemas Fracamente Acoplados

Sistemas de Computação independentes, mas conectados (multicomputadores)

Processamento Distribuído

SO de Rede (SOR) X SO Distribuído (SOD)



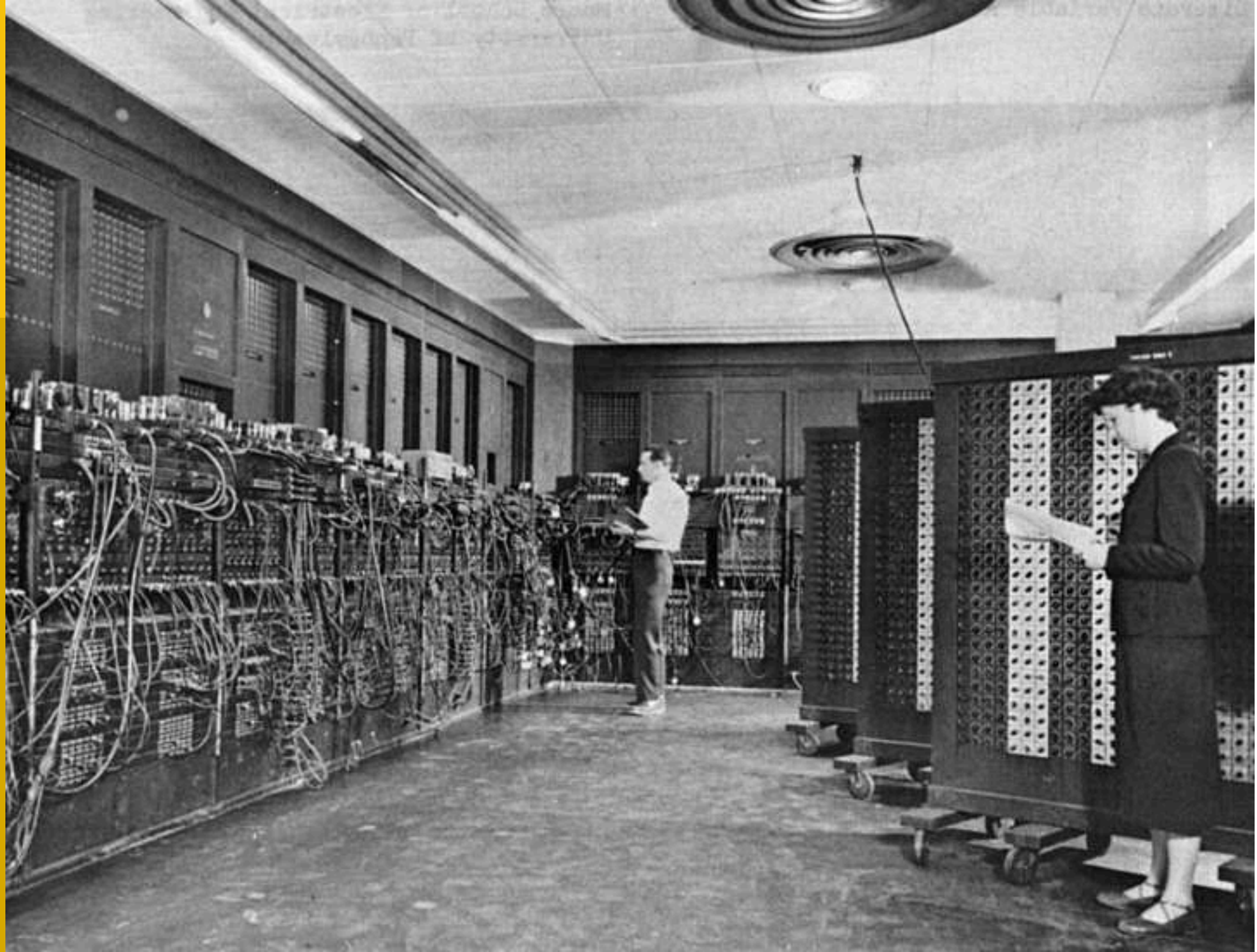
Tipos de Sistemas Operacionais

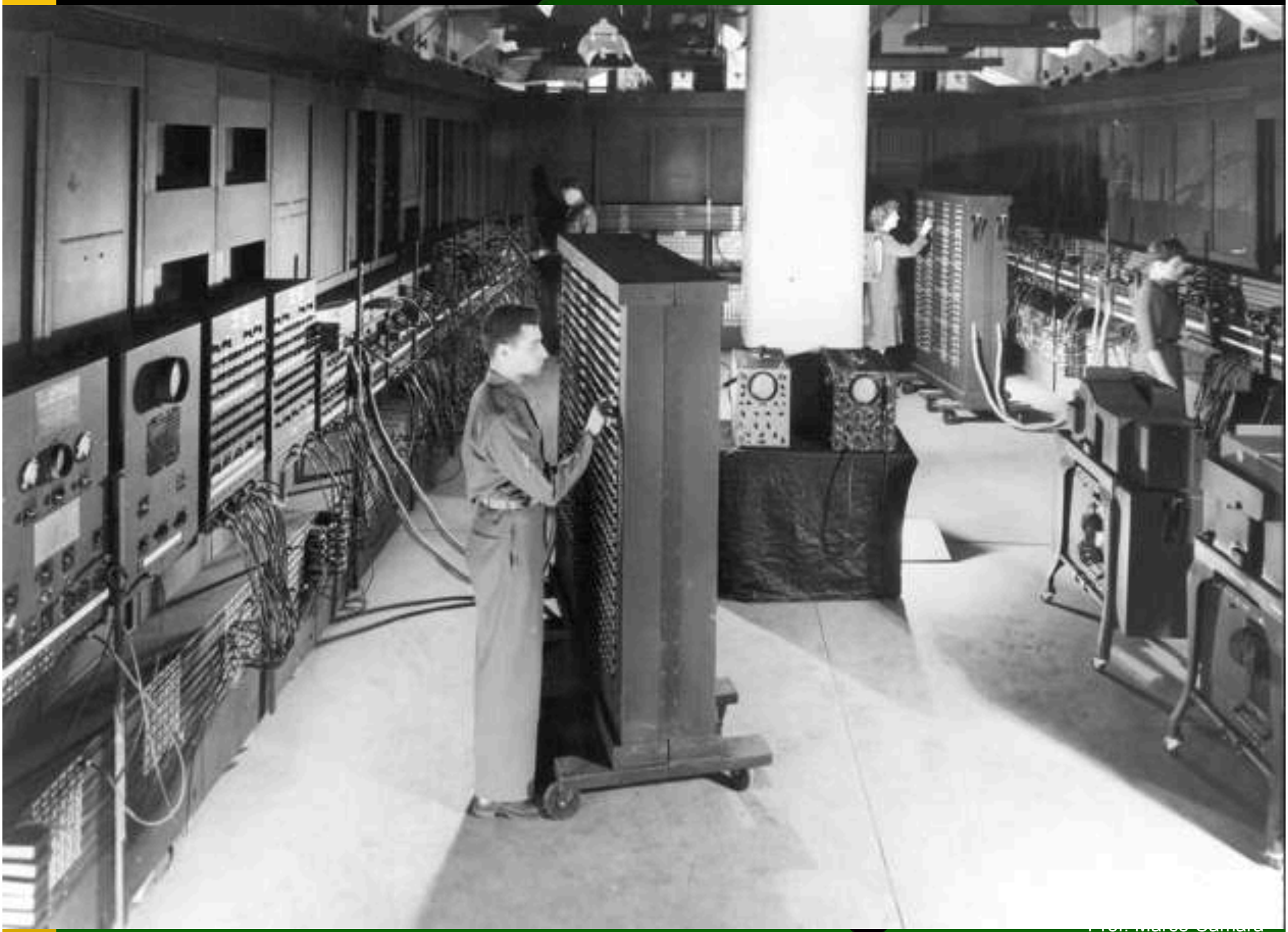
- Multiprocessamento
  - Computadores vistos originalmente como máquinas seqüenciais
    - Execução sequencial das instruções do programa
  - Sistemas Multiprocessados
    - Paralelismo - Simultaneidade
    - Execução de várias tarefas ou sub-tarefas

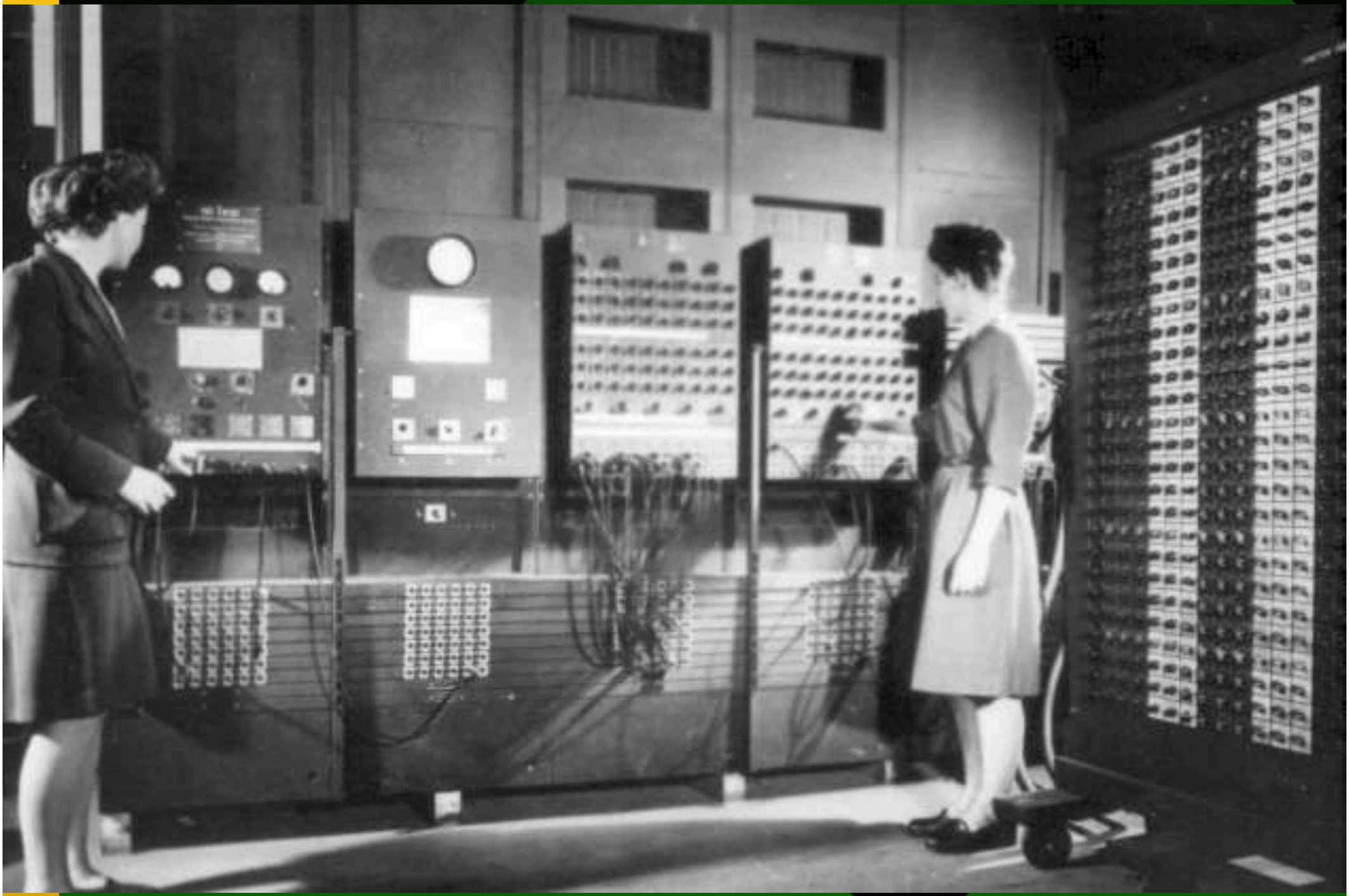
Histórico - 1ª Geração

(1945 - 1955)

- Primeiros computadores
- ENIAC (Electronic Numerical Integrator and Computer) - cálculos balísticos
- Utilização de válvulas (ENIAC possuía 18.000 válvulas)
- Programação em linguagem de máquina
- Não havia Sistema Operacional








THE ENIAC STORY
Celebrating six decades of electronic computing

arco Câmara

Histórico - 1ª Geração

(1945 - 1955)

- Outro exemplo: UNIVAC (*Universal Automatic Computer*)
  - Censo americano de 1950


Remington Rand


UNIVAC

Histórico - 2ª Geração

(1956 - 1965)

- Criação dos transistores
  - Aumento da confiabilidade (sem *bugs* ?)
  - Dimensão e consumo de energia bem menor
- Criação das memórias magnéticas
  - Acesso mais rápido aos dados
- Primeiras linguagens de programação:
  - Assembly e FORTRAN

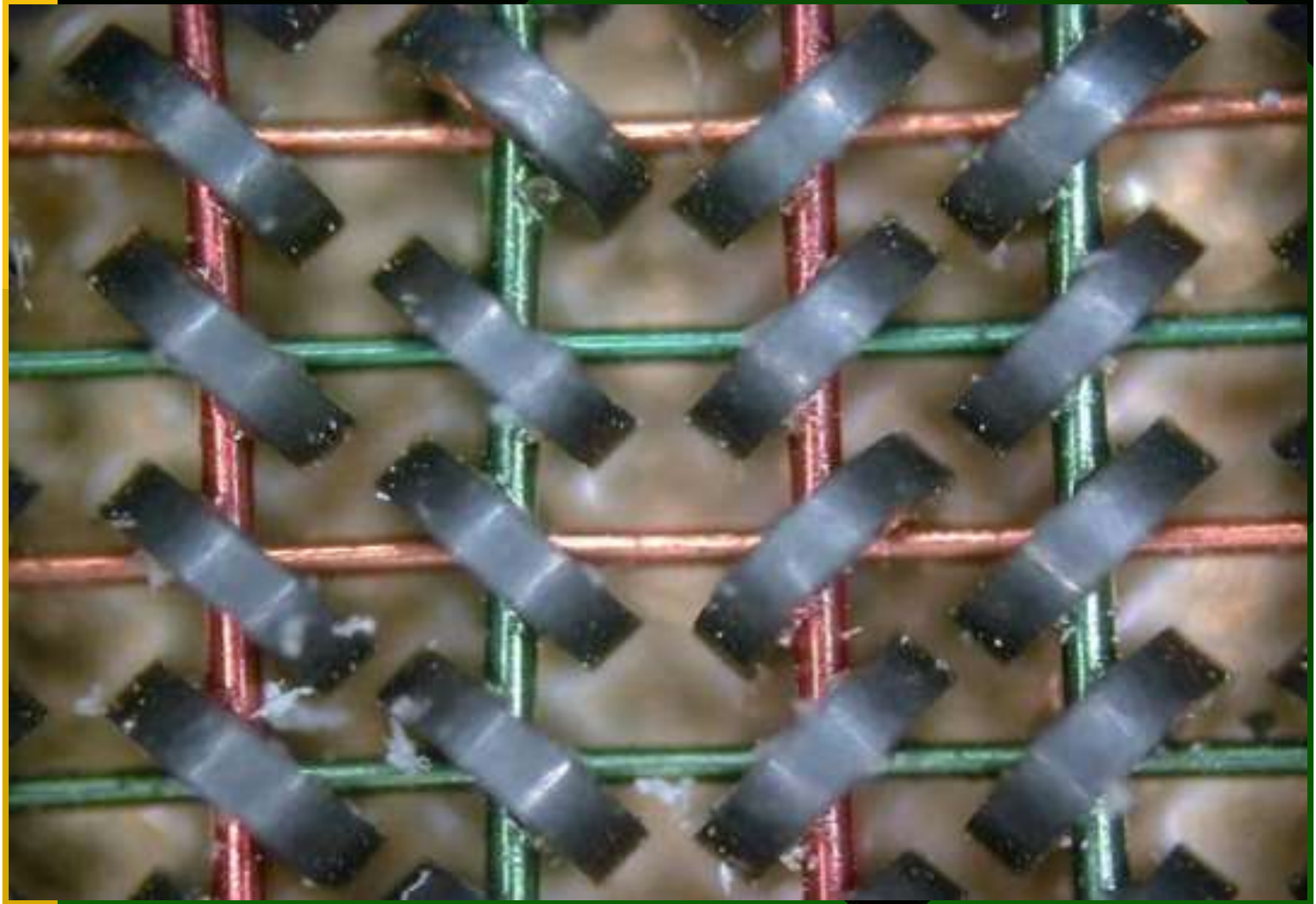
...ção

...*gs* ?)

...m menor

...icas

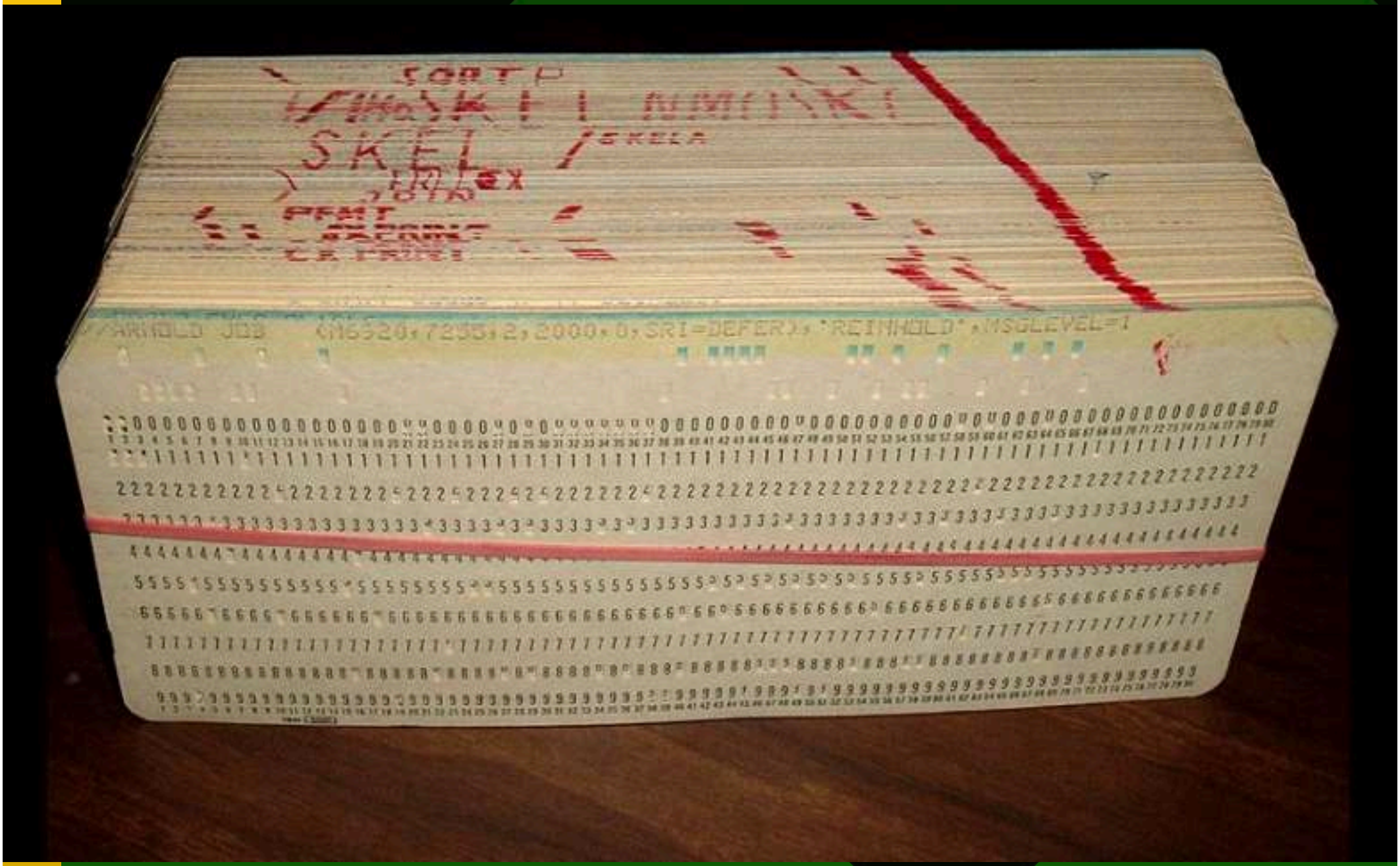
...amação:



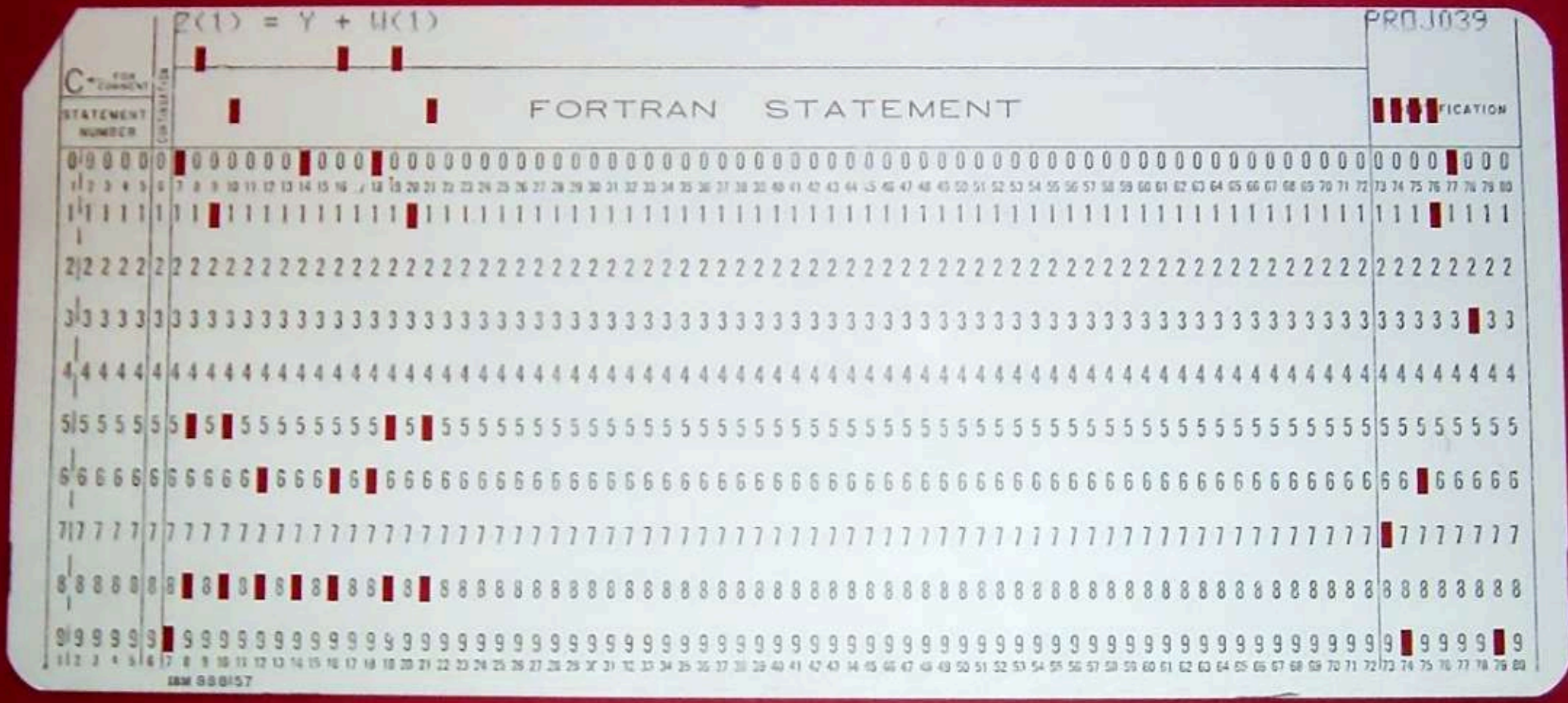
Histórico - 2ª Geração

(1956 - 1965)

- Cartão perfurado
- Processamento *batch*
- Rotinas para operações de Entrada e Saída (IOCS)
- Conceito de independência dos dispositivos


SORTP
SKEL
SKELA
PRNT
PRINT
//ARNOLD JOB (M6520,7235,2,2000,0,SRI=DEFER),'REINHOLD',MSGLEVEL=1

Histórico - 2ª Geração



Históricos - 2ª Geração

Histórico - 3ª Geração

(1966 - 1980)

- Monitor de vídeo e teclado (interação)
- *Time Sharing*
- Sistema Operacional UNIX
- Linguagem C
- Primeiros microcomputadores

Histórico - 3ª Geração

(1966 - 1980)

- Introdução dos Circuitos Integrados
  - Redução de custo e dimensões (+complexidade)
- Multitarefa / Multiprogramação
- Compartilhamento da memória
  - Primitivas com bloqueio
  - Sinais e interrupções

Histórico - 4ª Geração

(1981 - 1990)

- Aperfeiçoamento dos circuitos integrados
- Surgimento dos PC's e do DOS
- Estações de trabalho (monousuárias)
- Multiprocessadores
- Sistemas operacionais de rede e distribuídos.

Histórico - 5ª Geração ?

(1991 -)

- Arquitetura cliente-servidor
- Processamento distribuído
- Linguagem natural
- Segurança, gerência e desempenho do SO e da rede
- Consolidação dos sistemas de interfaces gráficas

Máquina de Níveis

Computador como máquina de níveis ou camadas:

- Nível 2 - Aplicações;
- Nível 1 – Sistema Operacional;
- Nível 0 – *Hardware.*

Nível 0 (*Hardware*)

Dispositivos Físicos;

Microprogramação;

Linguagem de Máquina.

Níveis 1 e 2 - *Software*

Sistema Operacional;

Linguagens de Programação

- Compiladores e Interpretadores;
- Máquinas Virtuais (Java);
- Ambientes de Desenvolvimento.

Aplicações.

Aula 03

Conceitos de *Hardware*

Definição de Processo

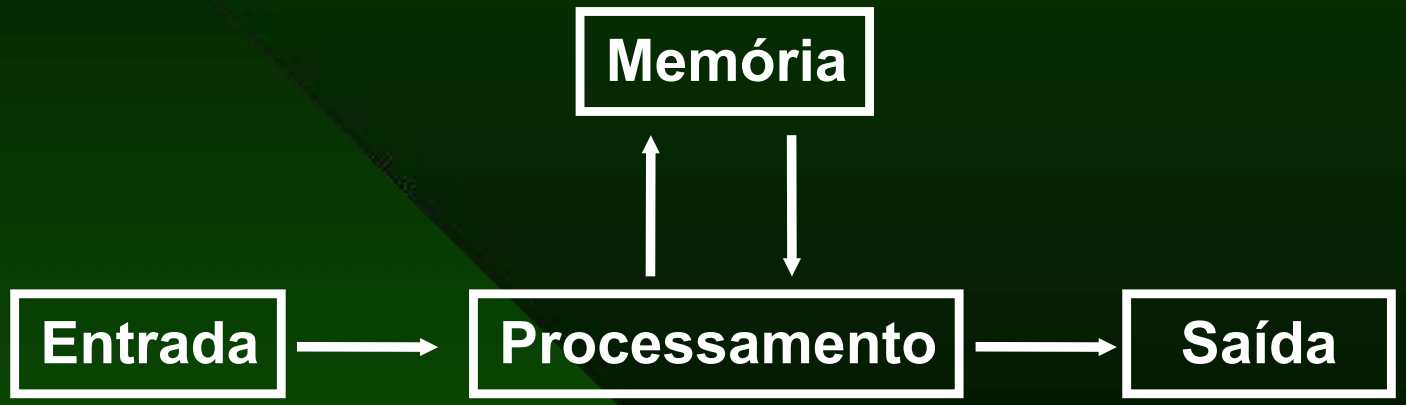
Escalonamento de processos - introdução

Conceitos

Hardware

- Três subsistemas básicos:
  - Unidade Central de Processamento;
  - Memória principal;
  - Dispositivos de entrada e saída.
- Subsistemas são também chamados de unidades funcionais;
- Implementações podem variar a depender da arquitetura.

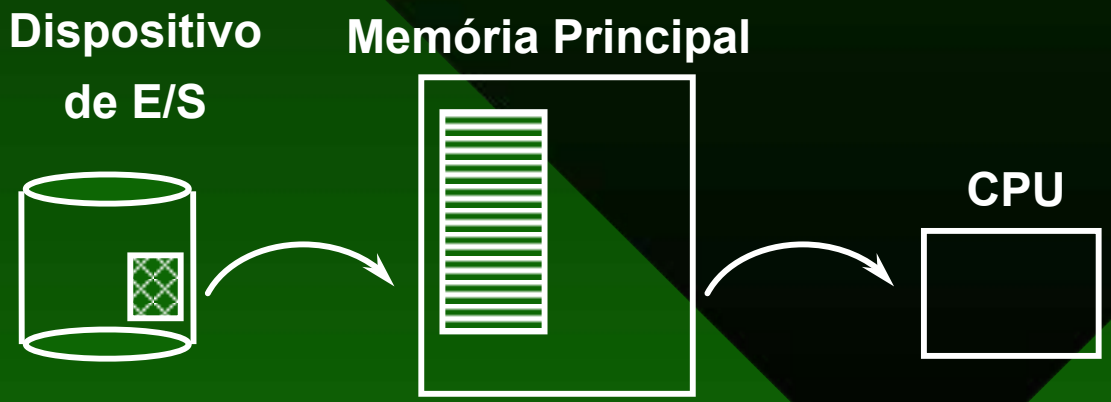
Modelo de Computador



Modelo de Computador

Execução de programas

- Programa armazenado em dispositivo de E/S
- Carga do programa na memória (SO)
- Execução das instruções (uma a uma) na CPU



CPU

Controla cada unidade funcional do sistema

Execução das instruções dos programas

- Aritméticas
- Comparação
- Movimentação

CPU

Conjunto de Instruções

- Instruções de máquinas disponíveis na CPU
- Tipos e quantidade variam de acordo com a CPU
- Compatibilidade de programas
- RISC x CISC

CPU

Desempenho de processadores

- Depende de:
  - Conjunto de instruções disponível
  - *clock*
  - Número de núcleos
- *Benchmark*
  - Usado para comparar processadores diferentes
  - Códigos específicos podem privilegiar um determinado produto !

Memória (definição)

- Externa (*storage*)
  - Armazena arquivos do SO, Aplicações e Dados;
  - Estante?
- Principal ou Interna
  - Provê o espaço de trabalho (mesa?)

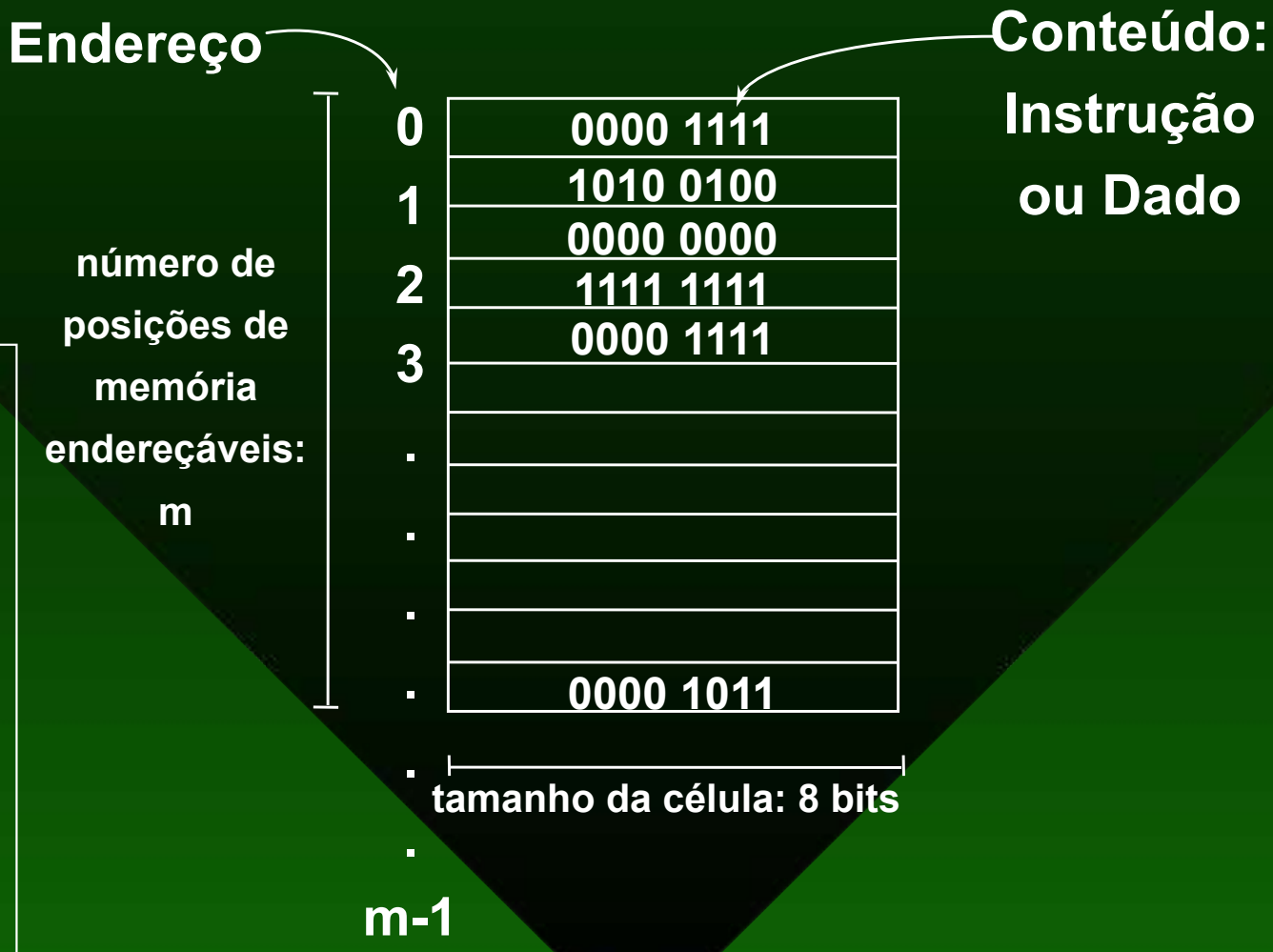
Memória Principal

- Armazena informações (*bits*)
  - Programas (instruções)
  - Dados
- Composta por várias unidades de acesso
  - Bit, Byte e Palavra;
  - Posições de memória.

Memória - Endereçamento

Cada posição de memória tem um endereço físico

Referência única de uma posição de memória



Memória - Tipos

Fisicamente, existem dois tipos:

- RAM, ou memória de leitura e escrita
  - Estática X Dinâmica;
  - A mais comumente especificada e manipulada;
  - Definida basicamente pela capacidade e barramento (performance).
- ROM
  - EPROM, EEPROM, Flash

Tipos de RAM

SRAM (*Static* RAM)

Construída com portas lógicas;

- Usa circuitos flip-flop para armazenar cada bit de memória, logo mantém as informações enquanto existir fonte de energia;
- Muito rápida;
- Cara e complexa (6 transistores), logo não é usada em grandes volumes.

DRAM (*Dynamic* RAM)

Construída c/componentes discretos

- Armazena cada bit em um capacitor conectado a UM transistor;
- O capacitor perde carga rapidamente, logo a memoria precisa ser “lembrada”;
- Simples e de baixo custo, compõe o maior volume da memorial usada nos computadores atuais.

Memória - Operação

Interligação Memória - Processador

Registradores de uso específico:

Memory Address Register - MAR - REM

Memory Data Register - MDR - RDM

Memória - Operação

Operação de Leitura da Memória

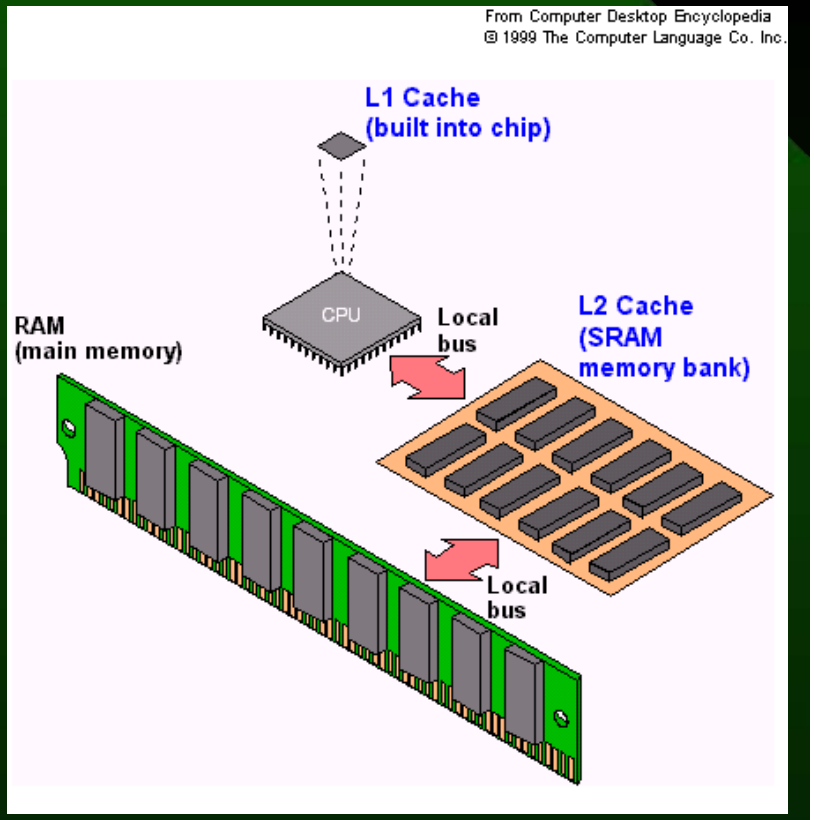
- CPU armazena endereço da célula a ser lida no MAR
- CPU gera sinal de controle indicando que a operação é de leitura da memória
- Memória recupera informação armazenada na posição endereçada e coloca no barramento de dados, chegando ao MDR

Memória - Operação

Operação de Gravação na Memória

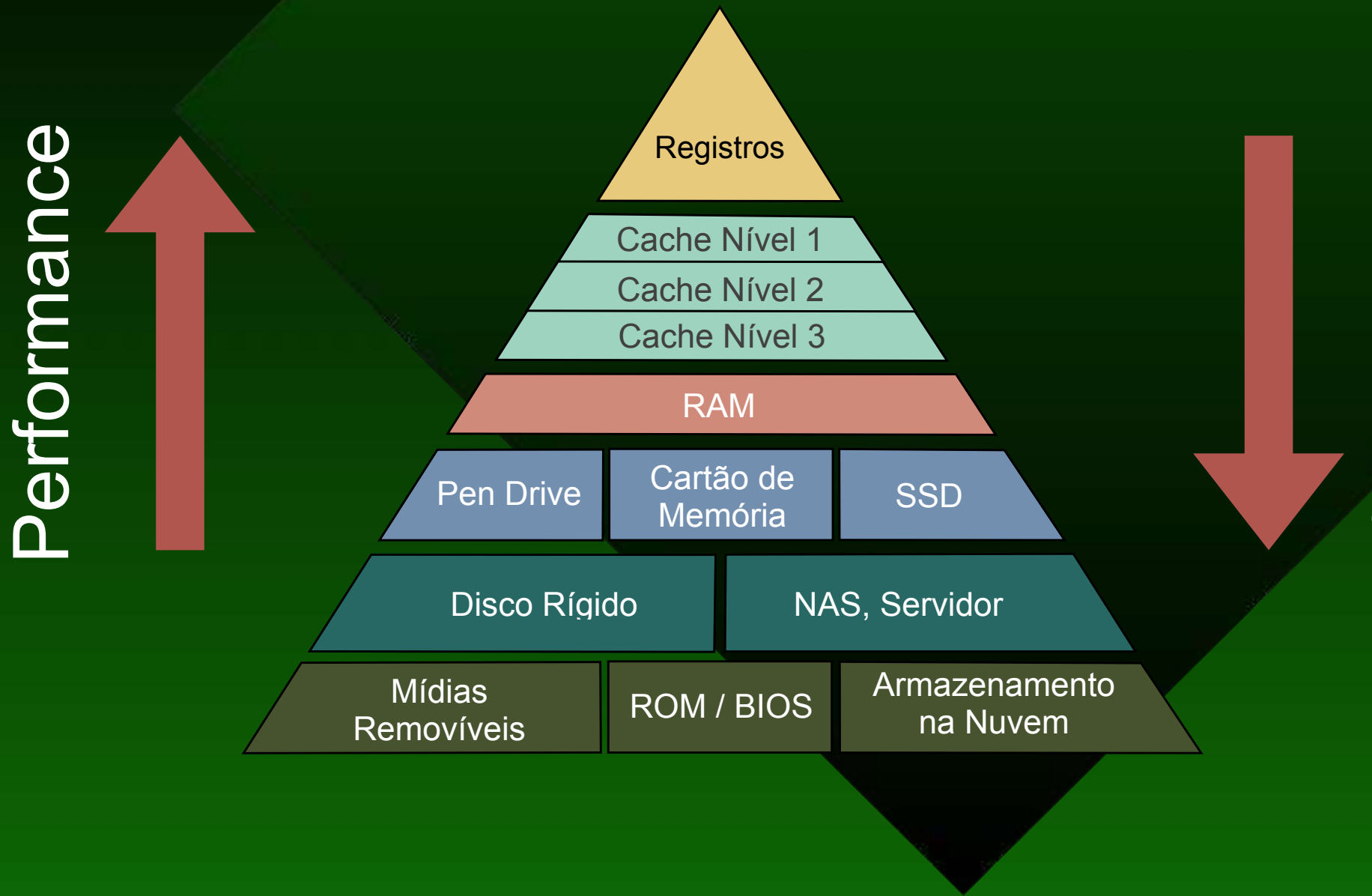
- CPU armazena endereço da célula a ser gravada no MAR
- CPU armazena a informação a ser gravada no MDR
- CPU gera sinal de controle indicando que a operação é de gravação na memória
- Memória armazena informação do barramento de dados na posição endereçada

Memória Cache



- Memória Cache
  - Memória de alta velocidade
  - Localizada entre CPU e Memória Principal
  - Aumento de desempenho com custo razoável
  - Algoritmo busca melhorar *Hit rate (*taxa de acertos)
  - Interna (L1,L2 … , ou primária) x Externa (Ln, ou secundária)

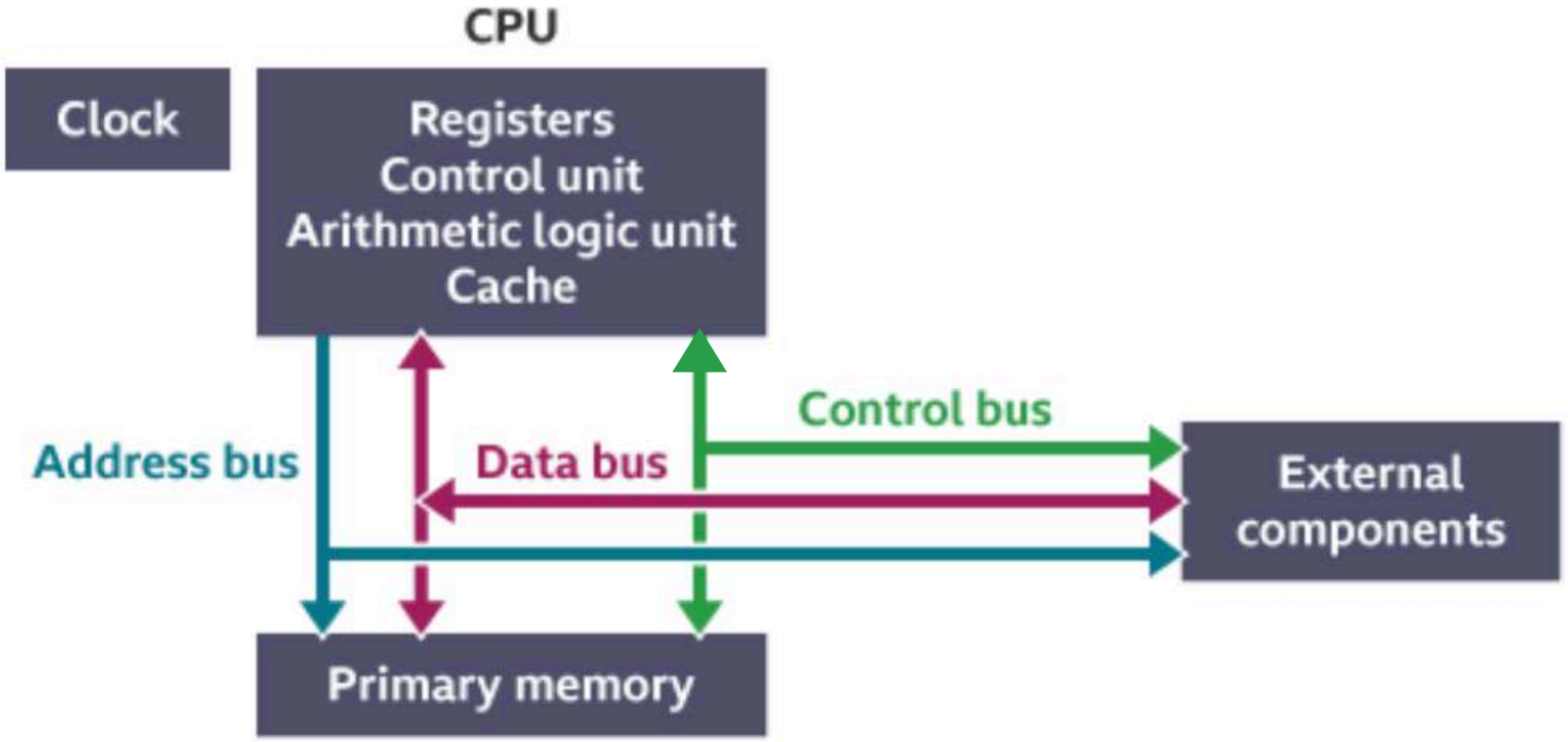
Hierarquia de Memória



Dispositivos de E/S

- Funções
  - Comunicação com meio externo
  - Memória Secundária e Interface Homem-Máquina
- Comunicação acontece através dos barramentos
  - Conjunto de vias paralelas para condução de sinais
  - Interligam CPU, Periféricos e Memória Principal

Barramentos



Barramentos

Barramento de Dados

- Número de *bits* por operação de E/S ou acesso à Memória
- Processadores de 8, 16, 32 ou 64 bits (palavra)
- Fluxo Bi-direcional (leitura / gravação)

Barramento de Endereços

- Capacidade de Armazenamento = 2^n
- Fluxo Uni-direcional (CPU ® MEM ou E/S)

Barramentos

- Barramento de Controle
  - Alguns exemplos de sinais de controle
    - *Clock;*
    - $\overline{Read}$ / *Write* ; $\overline{Write}$ / *Read;*
    - Controle de Interrupções;
  - Fluxo Uni ou Bi-direcional (depende do sinal de controle)

Definição de Processo

- Um programa em execução
- Não é o mesmo que Programa (entidade estática)
  - Programa é o código executável
  - Processo é o código executando
- Entidade dinâmica
  - Ex.: A execução de *prog.exe* 3 vezes gera 3 processos distintos do mesmo programa

Ambiente de um Processo

Todo processo precisa ter:

- Seção de texto
  - Código executável
- Seção de dados
  - Variáveis, estruturas
- Pilha do processo
  - Parâmetros etc
- Registradores, incluindo PC, SP, BP etc.

Conceitos

Para que os processos executem em ambiente multiprogramado, existe a gerência de:

- Compartilhamento da CPU, de memória, E/S etc

Nomenclatura

- Sistemas Batch
  - *Job*
- Sistemas de Tempo Compartilhado
  - Tarefa (*on-line*)

Estados do Processo

Mudanças de estado durante a execução:

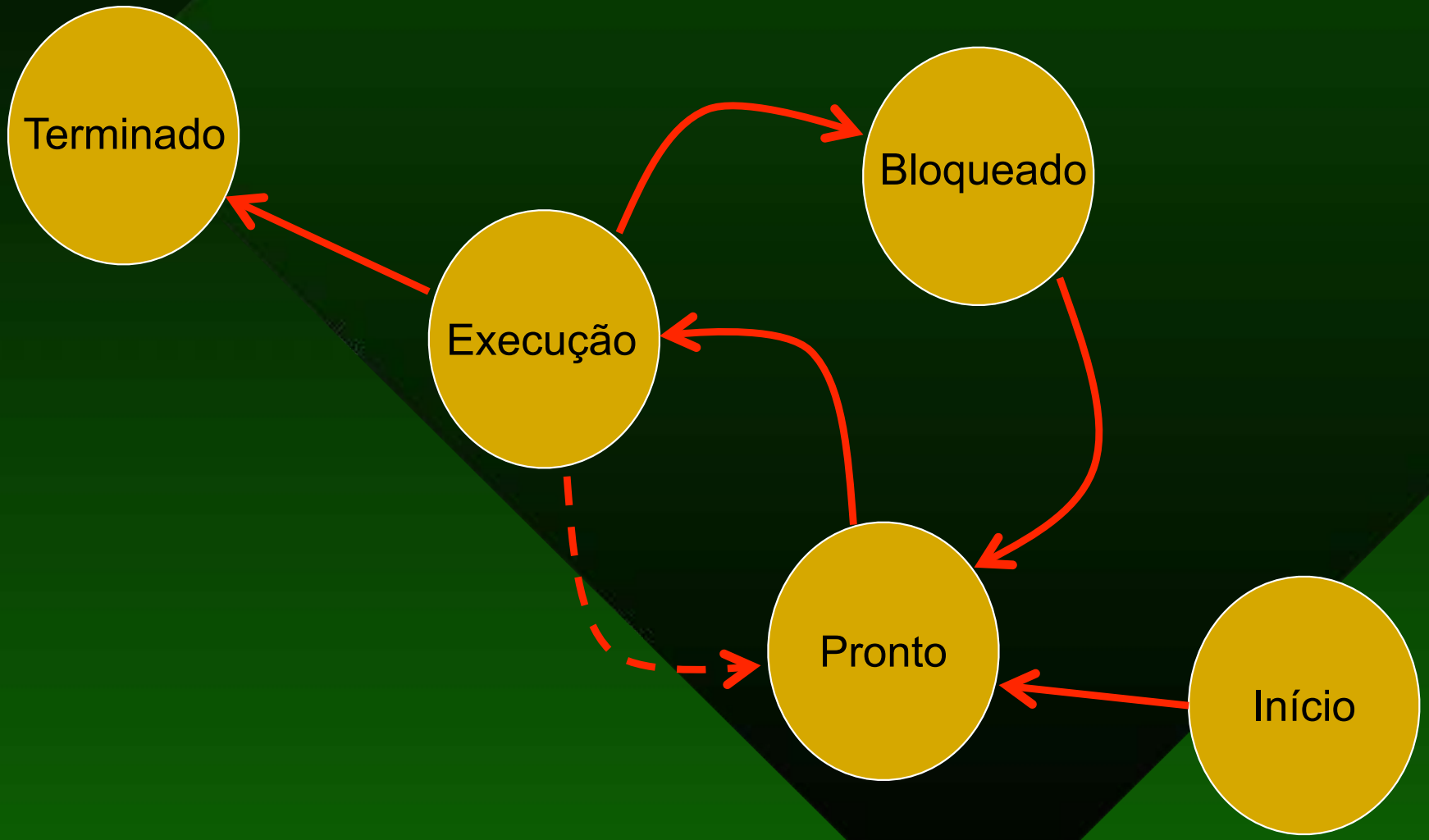
- Iniciando
  - O processo está sendo criado
    - SO Aloca Memória, Contexto (BCP)
      - Ex.: O usuário *clica* num arquivo executável no Explorer ou executa via CMD
- Executando
  - Instruções do processo estão sendo executadas
- Bloqueado
  - O processo está esperando algum evento externo

Estados do Processo

Mudanças de estado durante a execução:

- Pronto
  - O processo está aguardando chance de ser executado no processador
- Terminando
  - O processo está finalizando sua execução

Escalonamento de Processos



Aula 04

Escalonamento de Processos

Controle de Processo

- Bloco de Controle de Processo (BCP)
  - Área de Memória alocada pelo SO para gerenciar processos
  - Mantém informações sobre o processo
    - Estado do processo
    - Registradores da CPU, incluindo o PC
    - Informações para escalonamento
    - Informações para gerenciamento de memória
    - Informações de contabilização
    - Informações sobre operações de E/S
    - Ponteiros para Arquivos, *Socket* etc

Troca de Contexto

- Preempção
  - Ação de retirar um processo da CPU a qualquer instante e restaurá-lo como se nada tivesse ocorrido
    - Suportado por Interrupções (como pelo CLOCK) e pelo BCP
      - Ex.: Preempção por tempo de Quantum
        Preempção por I/O, Prioridade etc.
- Tempo da troca é considerado *overhead*
- Tempo depende de suporte de *hardware* e complexidade do SO

Escalonamento de Processos

- Manutenção de filas para controle dos processos
  - Fila de Processos – todos processos no sistema
  - Fila de Pronto – processos prontos
  - Filas de Bloqueado - processos aguardando E/S de dispositivo (interrupção) ou sinal
    - Múltiplas Filas – uma por dispositivo
      - Fila de Disco, Fila de CD, Fila de Teclado, de Rede
  - BCP efetua o encadeamento nas filas
  - Processos passam por várias filas

Escalonadores

Classificação de processos:

- I/O-BOUND – Intensivamente consumidor de E/S
  - Passa mais tempo fazendo operações de E/S do que utilizando a CPU
    - Processos Comerciais
- CPU-BOUND – Intensivamente consumidor de CPU
  - Passa mais tempo efetuando cálculos do que E/S
    - Processos Científicos/Matemáticos
- Híbrido

Escalonadores

Combinação adequada de processos CPU e I/O BOUND melhora o desempenho e flexibilidade do sistema

- Processo interativo (Ex.: Internet Explorer)
  - Tipicamente I/O Bound

Algoritmo de Escalonamento de CPU

Algoritmo do S.O. que determina qual o próximo processo a ocupar a CPU

- Executado quando ocorre estouro de Quantum ou interrupção do processo (I/O, Evento, Sinal etc.) ou o processo acaba;

Critérios mudam com características dos Processos

- *Batch, CPU Bound, I/O Bound*, Interativos

Metas do Escalonamento

Eficiência

- Manter a CPU ocupada 100% do tempo

Throughput

- Maximizar o número de processos (tarefas, jobs) executados em um dado intervalo de tempo

Turnaround

- Minimizar o tempo de um processo no sistema, desde seu início até o término
  - Tempo médio de execução
  - Fundamental a processos Batch

Metas do Escalonamento

Igualdade

Todo Processo tem direito de ocupar a CPU

Tempo de resposta

Minimizar o tempo decorrido entre a submissão de um pedido e a resposta produzida num processo interativo

Conflito entra Metas

- Atender a uma meta pode prejudicar outra
  - Qualquer algoritmo de escalonamento favorecerá um tipo de processo (*CPU Bound, I/O Bound*, Tempo Real, etc) em detrimento de outros
  - O propósito precisa ser geral

Tipos de Escalonamento

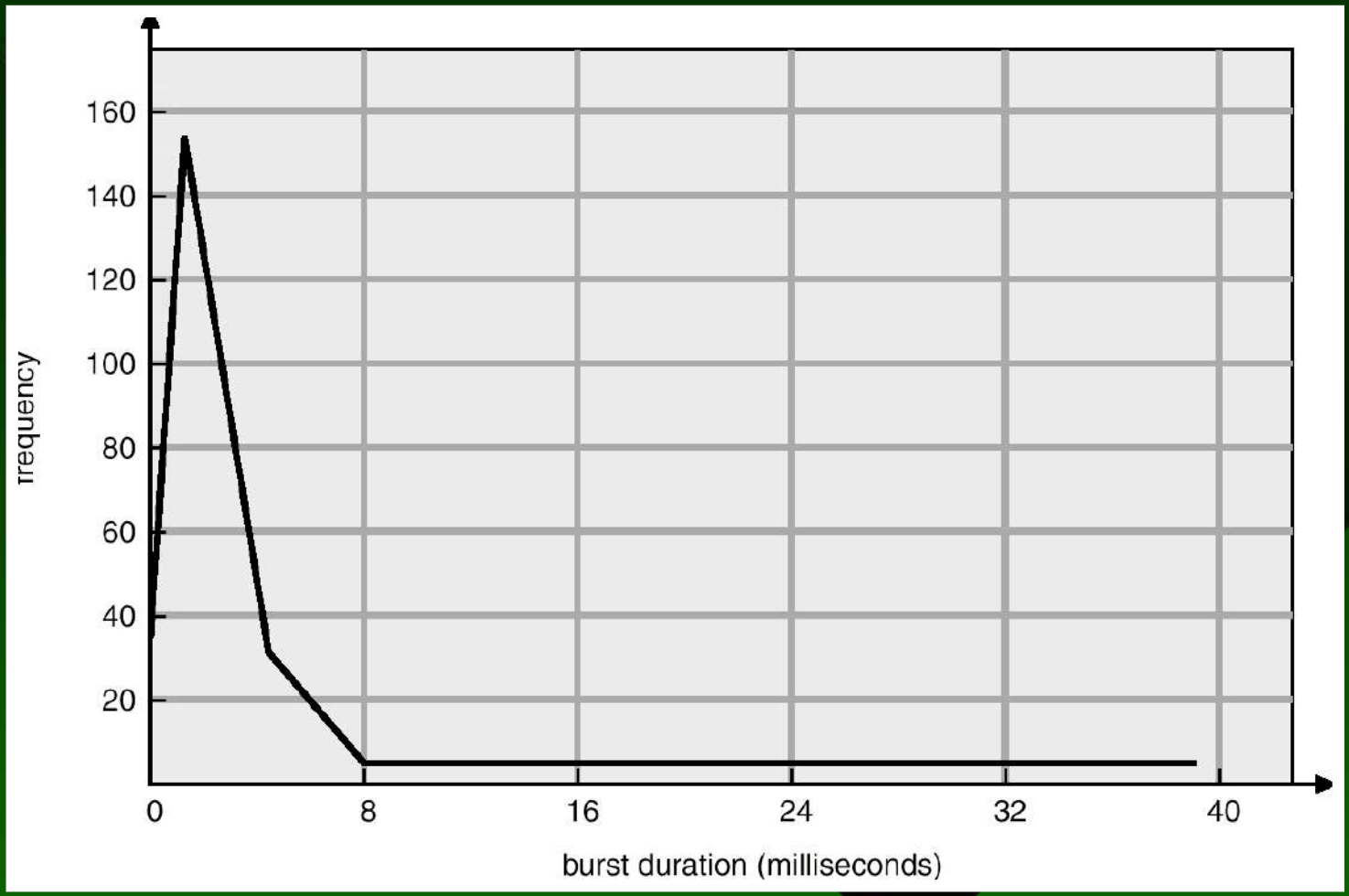
Escalonamento não-preemptivo

- Escalonamento Cooperativo;
- Processo mantém a CPU até terminar ou E/S;
- Não requer recursos especiais de hardware
  - Não existe Quantum (devolução voluntária do controle ao S.O.)
- No caso do Windows, utilizado até a versão 3.x.

Escalonamento preemptivo

- Requer temporizador na CPU (fatia de *quantum* ou Uso do *clock*);
- Requer suporte do SO para coordenar acesso a dados compartilhados de forma consistente (proteção).

Frequência de processos por duração de surto de CPU



Escalonamento FIFO

First Come First Served (FCFS, FIFO, PEPS)

Não preemptivo

Processo	Início	Duração (ut)
P_1	0	24
P_2	0	3
P_3	0	3

Escalonamento FIFO

Ordem de chegada dos processos:

$$P_1, P_2, P_3$$

Diagrama de Gantt

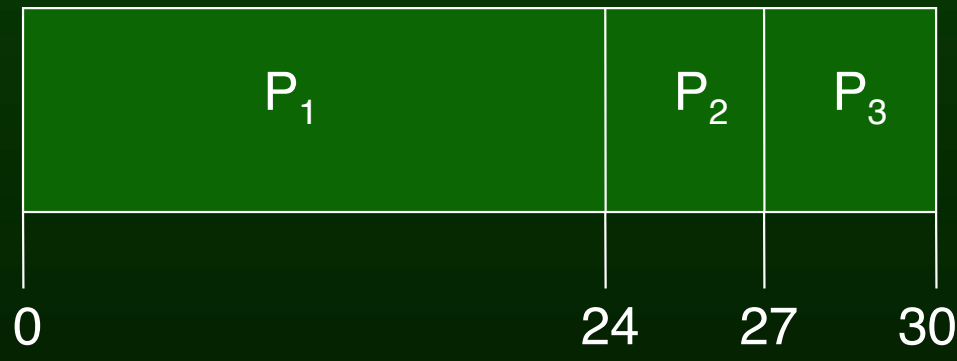
Escalonamento FIFO

Tempos de espera

$P_1 = 0$

$P_2 = 24$

$P_3 = 27$



Throughput = 0,1 (3/30)

Dica: Tempo de Espera = tempo no estado de Pronto.

Tempo médio de espera

(0 + 24 + 27) / 3 = 17

Escalonamento FIFO

Tempos de saída

$P_1 = 24$

$P_2 = 27$

$P_3 = 30$

Tempo médio de saída

(24 + 27 + 30) / 3 = 27

Escalonamento SJF

Shortest-Job-First (Menor Job Primeiro)

- Melhor: “próximo surto de CPU menor primeiro”

Usado para Processos *batch*.

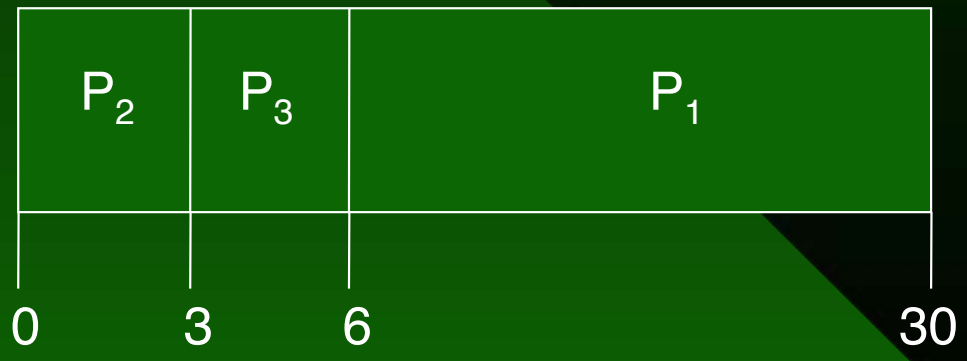
Execução diária permite determinar seu tempo total.

Escalonamento SJF (no mesmo exemplo)

Outra ordem de chegada

P_2 , P_3 , P_1

Diagrama de Gantt

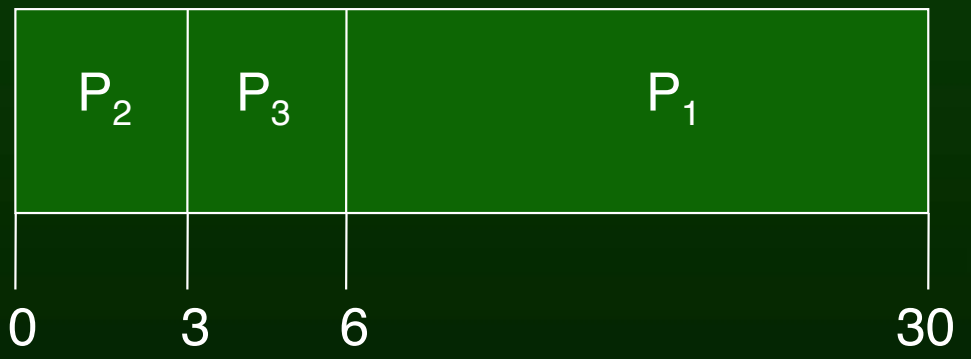


Escalonamento SJF (no mesmo exemplo)

FIFO ordenado (SJF / MPP)

Menor Processo Primeiro

Menor tempo de execução



Tempos de espera

$TEP_1 = 6$; $TEP_2 = 0$; $TEP_3 = 3$

Tempo médio de espera melhora

(6 + 0 + 3) / 3 = **3** (era **17** no FIFO)

Tempo médio de espera não é mínimo

Pode variar muito (com os surtos de CPU)

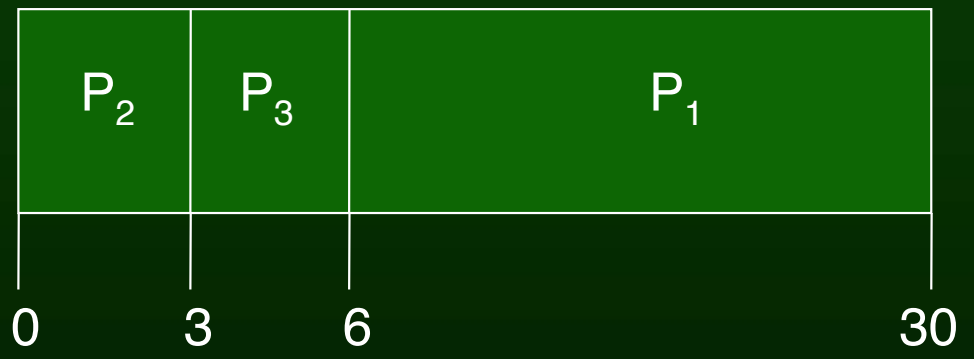
Efeito Comboio

Processos I/O bound esperam por CPU bound

Escalonamento SJF (no mesmo exemplo)

- Tempos de saída

  $P_1 = 30$; $P_2 = 3$; $P_3 = 6$



- Tempo médio de saída melhora

  (30 + 3 + 6) / 3 = **13** (era **27** no FIFO)

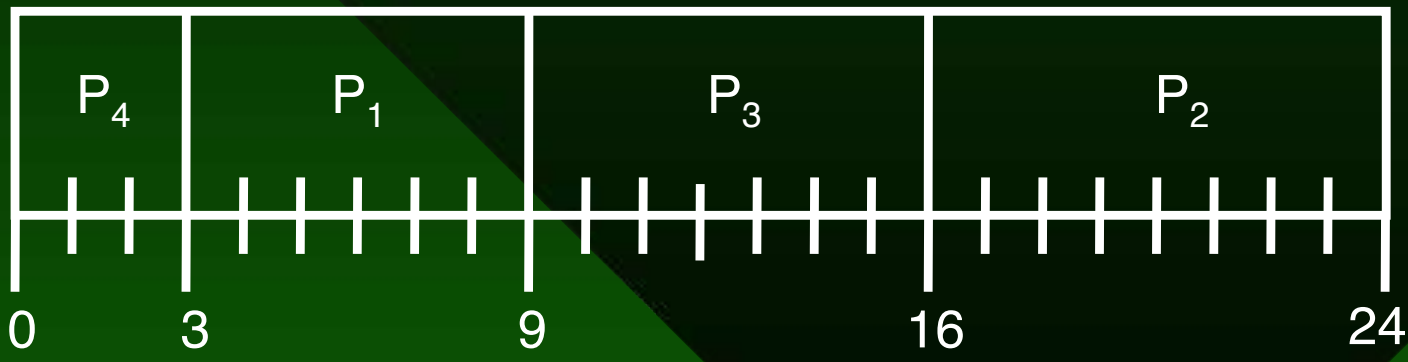
- Tempo médio de saída não é mínimo
  - Pode variar muito (com os surtos de CPU)

Escalonamento SJF (outro exemplo)

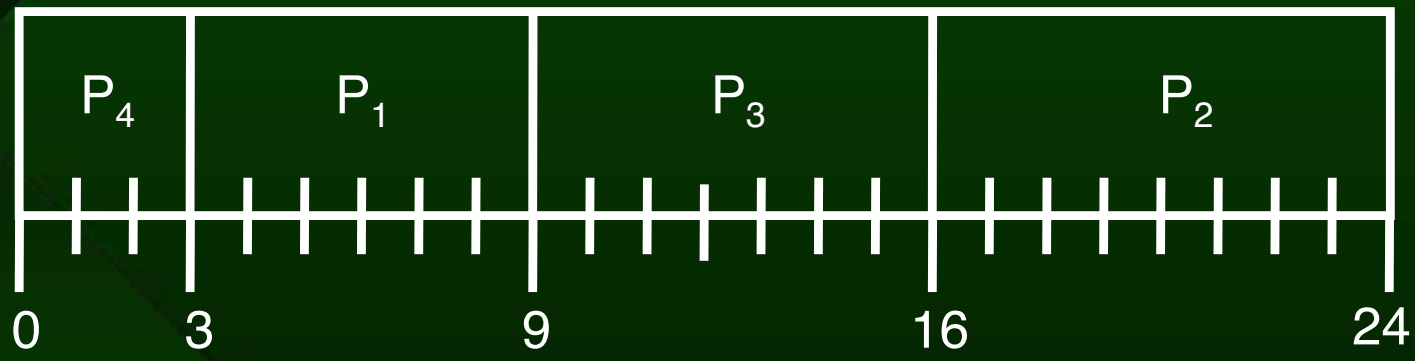
PID	Início	Duração surto
P_1	0	6
P_2	0	8
P_3	0	7
P_4	0	3

Tempos de espera

$P_1 = 3$; $P_2 = 16$; $P_3 = 9$; $P_4 = 0$



Escalonamento SJF (outro exemplo)



Tempo médio de espera melhora

(3 + 16 + 9 + 0) / 4 = 7

Para FIFO, nesta situação, seria 10,25 = (0+ 6+14+21)/4

Tempo médio de espera é mínimo

Algoritmo considerado ***ótimo***

Escalonamento SJF

- Problema: determinação exata da duração do próximo surto de CPU é impossível
  - SJF é usado para escalonamento de jobs em sistemas batch
    - Usuário especifica o tempo de CPU do job
  - Em escalonamento de CPU é usada estimativa
    - Baseada na duração dos surtos anteriores
      - Média exponencial

SJF com preempção

- Não preemptivo
  - Processo usa CPU até completar surto
- Preemptivo
  - Novo processo pronto com surto previsto (T_A)
  - Tempo restante previsto para o processo em execução (T_B)
  - Se $T_A < T_B \Rightarrow$ preempção por prioridade
  - *Shortest-Remaining-Time-First* (SRTF)

SJF com preempção

Processo	Instante de chegada	Duração de surto
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

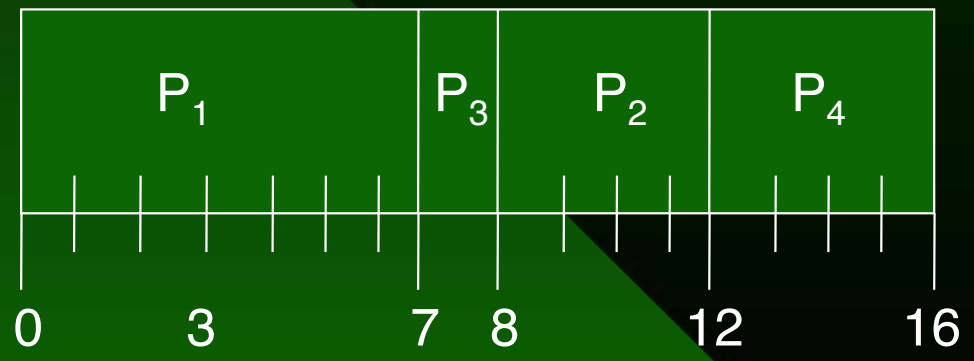
SJF com preempção

SJF não preemptivo

Tempo de espera médio = (0 + 6 + 3 + 7) / 4 = 4

$TEP_1 = 0$ $TEP_2 = 6$ (8 - 2)

$TEP_3 = 3$ $TEP_4 = 7$



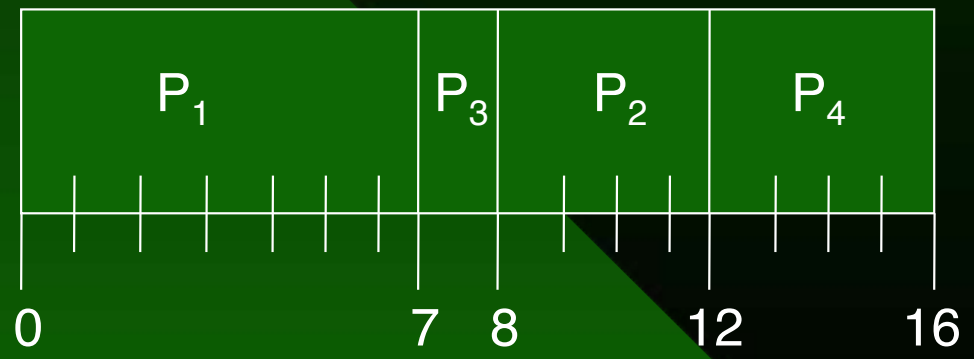
SJF com preempção

SJF não preemptivo

Tempo de saída médio = (7 + 10 +4 + 11) / 4 = 8

$TSP_1 = 7$ $TSP_2 = 10$

$TSP_3 = 4$ $TSP_4 = 11$



Tempo	Pronto	Execução	Term.	Observações
0	P1 (7)			
0		P1 (7)		
2	P2 (4)	P1 (5)		t(P2) < t (P1)
2	P1 (5)	P2 (4)		
4	P1(5), P3(1)	P2 (2)		t(P3) < t(P2)
4	P1(5), P2(2)	P3 (1)		
5	P1(5), P2(2), P4(4)		P3	
5	P1(5), P4(4)	P2 (2)		
7	P1(5), P4(4)		P2	
7	P1(5)	P4 (4)		
11	P1(5)		P4	
11		P1 (5)		
16			P1	

Proc.	Cheg.	Surto
P1	0	7
P2	2	4
P3	4	1
P4	5	4

SJF Preemptivo

Qual o tempo médio de espera ?

Qual o tempo médio de saída ?

SJF Preemptivo

Qual o tempo médio de espera ?

P1=0; P2=0; P3=0; P4=2

Média = 0,5

Qual o tempo médio de saída ?

P1=16; P2=5 (7-2); P3=1 (5-4); P4=6 (11-5);

Média = 7

Aula 05

Escalonamento de Processos (Tempo Real)

Escalonamento de Tempo Real

- Sistemas de tempo real crítico
  - Limites rígidos de tempo
  - SO garante execução no tempo ou rejeita
  - Exige software especial e hardware dedicado

Escalonamento de Tempo Real

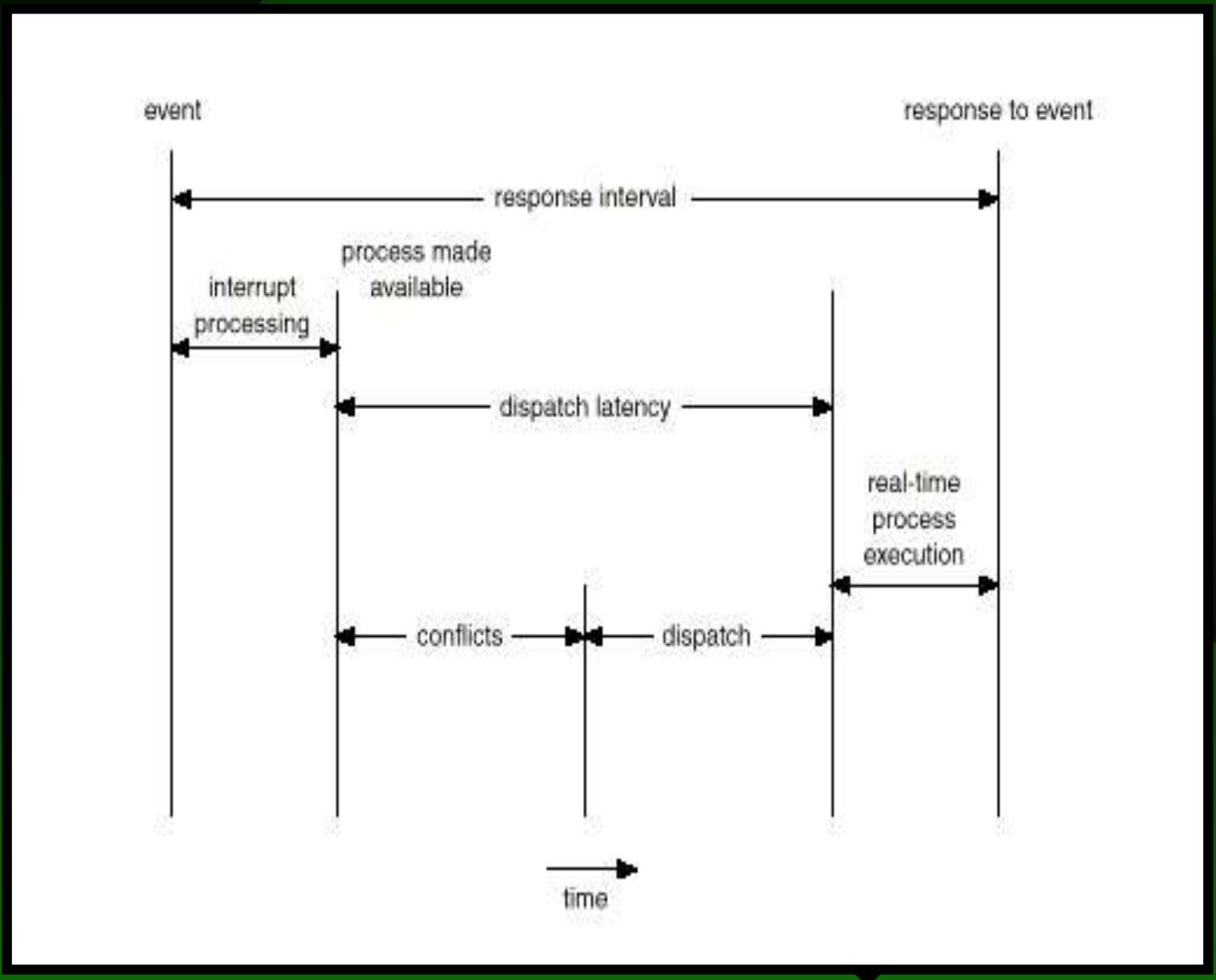
- Sistemas de tempo real não-crítico
  - Processos críticos com prioridade
    - Gera desbalanceamento do sistema
  - Suporte do SO
    - Escalonamento com prioridade
    - Não degradação da prioridade dos processos críticos
    - Latência de carga pequena
      - Chamadas ao sistema e operações de E/S
      - Pontos de preempção seguros
      - Todo Kernel preemptível (sincronização)
      - Protocolo de herança de prioridade

Escalonamento de Tempo Real

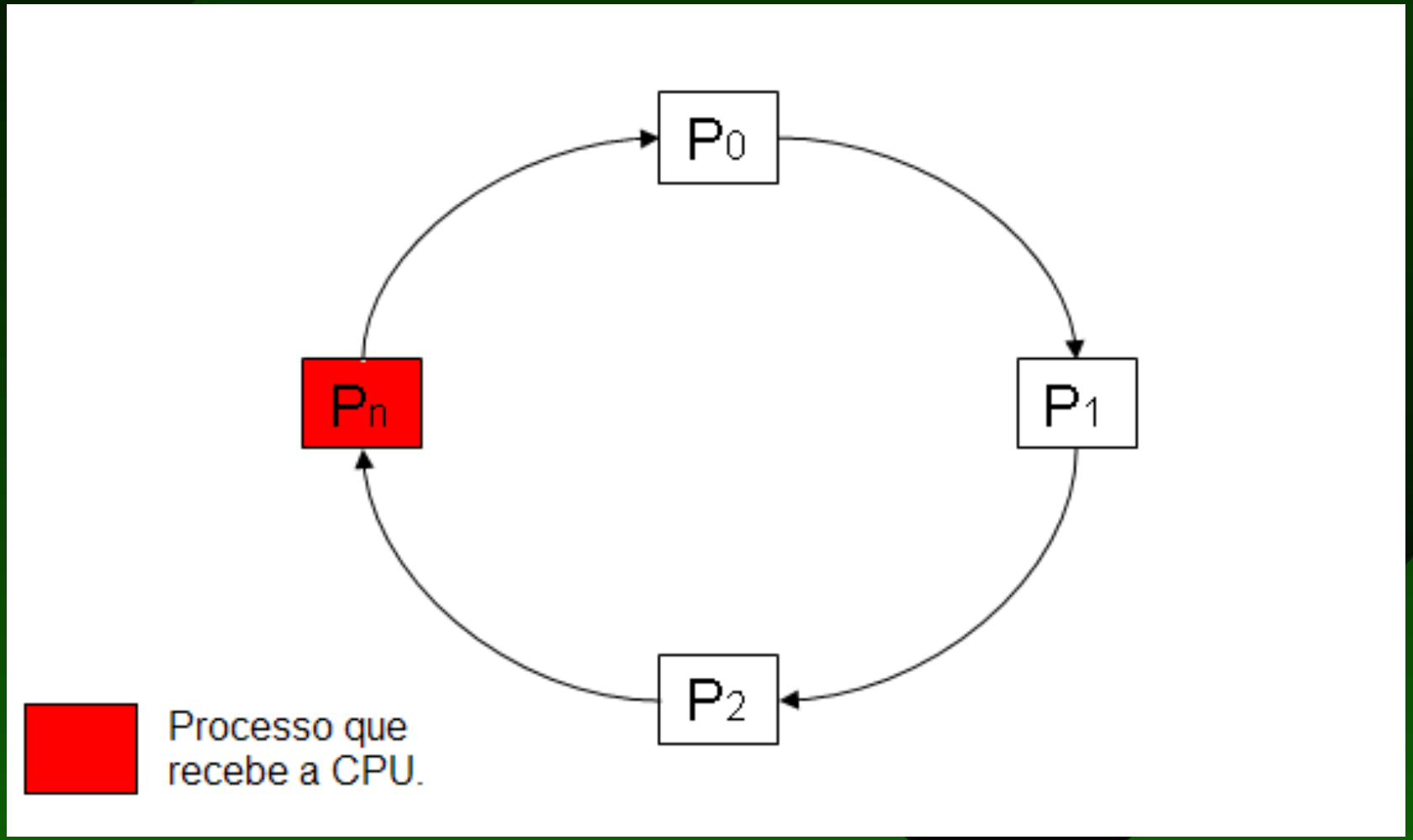
Dispatch Latency

- Descreve a quantidade de tempo que um sistema gasta para responder à requisição de um processo
- O tempo de resposta (TR) total consiste em:
  - TR de Interrupção
  - Dispatch latency
  - TR da aplicação

Escalonamento de Tempo Real



Escalonamento *Round Robin*



Escalonamento *Round Robin*

Round-Robin (revezamento circular)

- Método antigo e muito simples;
- Cada processo recebe uma fatia de tempo (Quantum), e é escalonado seguindo a ordem de uma fila circular;
- Sistema Preemptivo - Interrupção do Clock
- Se o Quantum acabar antes do término do processo (preempção), ou se o mesmo for bloqueado, o processo vai para o final da fila.

Escalonamento *Round Robin*

Resultados típicos:

- Elimina problemas de *starvation*;
- Tempo de espera médio é longo;
- Tempo de saída maior que SJF;
- Tempo de resposta melhor que SJF;
- Quantum grande -> FIFO;
- Quantum pequeno -> Baixa eficiência devido ao excesso de trocas de contexto.

Escalonamento *Round Robin*

- Análise da eficiência
  - Com quantum **q** e **n+1** processos prontos:
    - Tempo máximo de espera: n*q
- Um exemplo
  - Suponha uma fila de pronto com 100 processos, Quantum de 100 ms (valor típico);
  - Um processo interativo executa, faz uma requisição, vai para bloqueado e de lá para o fim da fila
    - Quando a resposta será entregue ao usuário do processo interativo?

Escalonamento por Prioridade

Cada processo tem uma prioridade

- Número inteiro dentro de limites (0 a 7 / 0 a 4095)
- Menor (ou maior) número ⇒ maior prioridade
  - Empate ⇒ Round Robin, FCFS

Starvation – Estagnação

- Bloqueio por tempo indefinido
- Soluções:
  - Decrementa prioridade a cada ciclo em execução;
  - *aging* (envelhecimento)

Escalonamento por Prioridade

Prioridade

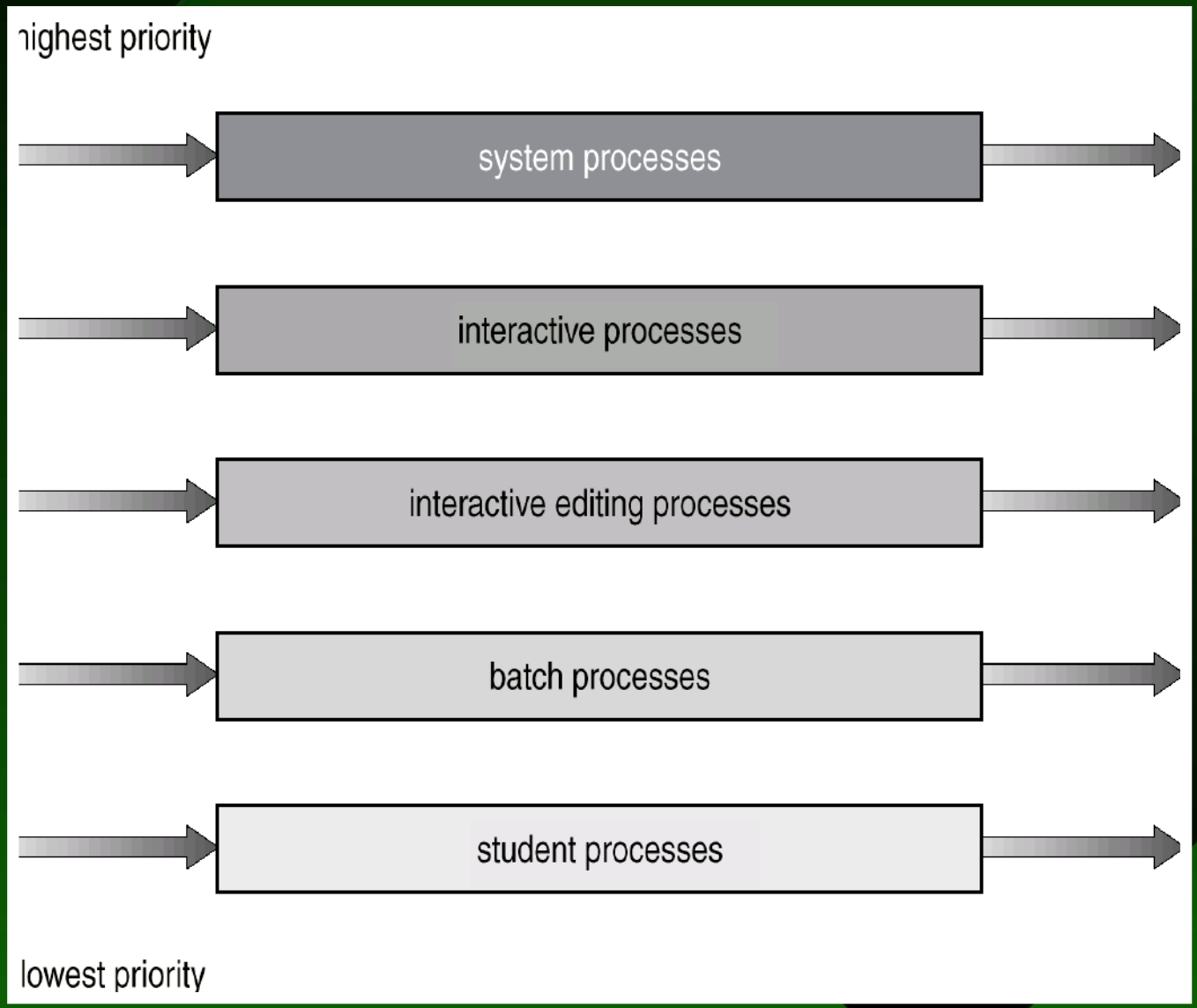
- Definida interna ou externamente;
- Estática ou Dinâmica;
- Alguns SO (Unix) permitem definir prioridade a um processo (comando *nice*);
- SO pode, por exemplo, aumentar prioridade de processos *I/O Bound*;
  - Ex: Prioridade = 1 / fração Quantum utilizada
- SJF: caso especial de prioridade ?

Pode-se classificar processos por classes de prioridade, e, dentro de cada classe, usar outro algoritmo de

Escalonamento por Múltiplas Filas

- Escalonamento preemptivo entre filas
  - Prioridade fixa: só atende filas menos prioritárias se as demais estiverem vazias
  - *Time slice* 80% para *foreground* com RR e 20% para *background* com FIFO

Escalonamento por Múltiplas Filas



Escalonamento por Múltiplas Filas

- Filas caracterizadas pelos surtos de CPU dos processos
  - *I/O bound* e interativos com mais prioridade
    - Passam a maior parte do tempo Bloqueados
  - Processos podem mudar de fila
  - *Aging* pode ser facilmente implementado
- Algoritmo preemptivo

Escalonamento com Múltiplos Processadores

- Escalonamento de CPU mais complexo
  - Existem sistemas com barramento de E/S privativo de determinado processador
- Várias filas de processos prontos
  - Possibilidade de desperdício de recursos

Escalonamento com Múltiplos Processadores

- Única fila de processos prontos
  - *Symmetric Multiprocessing* (SMP)
    - Cada processador faz seu escalonamento
    - Compartilhamento de estruturas de dados do SO
      - Sincronização
  - *Assymmetric Multiprocessing*
    - Escalonamento no processador mestre

Sincronismo de Processos

Sincronismo de Processos

É um problema inerente à gestão de recursos;

- Acesso simultâneo a recursos que precisam ser compartilhados;
- Pode gerar *dead-lock*, *starvation* ou inconsistência.

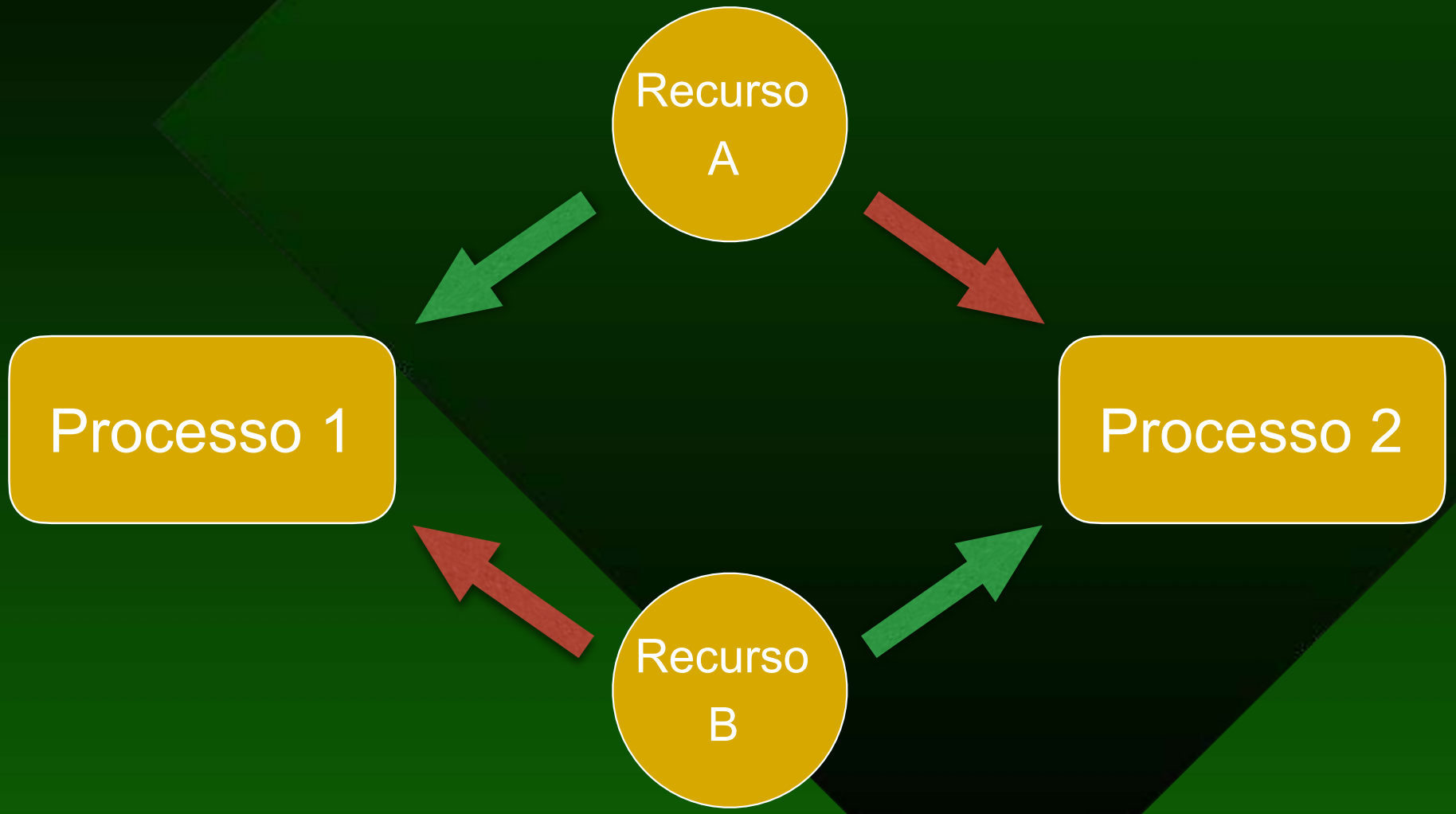
Deadlock

Sincronização de Processos

Dead-Lock

- Processo P1 bloqueia recurso R1 para seu uso;
- Processo P2 bloqueia recurso R2 para seu uso, e é bloqueado aguardando liberação de R1;
- Processo P1 é bloqueado aguardando R2.

Deadlock



Sincronização de Processos

Starvation

Política de escalonamento isola um ou mais processos, que nunca são escalonados, ou nunca conseguem ter acesso aos recursos necessários;

É necessário estudar algoritmos específicos para tratar este tipo de falha.

Sincronização de Processos

Inconsistência:

- Dois ou mais processos acessam a mesma base de dados, e alteram dados simultaneamente;
- Exemplos:
  - Processo A totaliza o campo X de todos os registros, enquanto que o Processo B altera o conteúdo de registros já contabilizados pelo Processo A -> Soma inconsistente?
  - Processos que manipulam informações de forma coordenada são interrompidos no meio de uma atualização. Dados inconsistentes?

Sincronização de Processos

Problemas clássicos

- Produtor / Consumidor;
- O jantar dos filósofos.

Sincronização de Processos

- Produtor / Consumidor
  - Dois processos compartilham um *buffer* de tamanho limitado
    - O processo produtor:
      - Produz um dado
      - Insere no *buffer*
      - Volta a gerar um dado
    - O processo consumidor:
      - Consome um dado do buffer (um por vez)

Sincronização de Processos

Sincronização de Processos

Produtor / Consumidor

O problema é:

Como garantir que o produtor não adicionará dados no *buffer* se este estiver cheio?

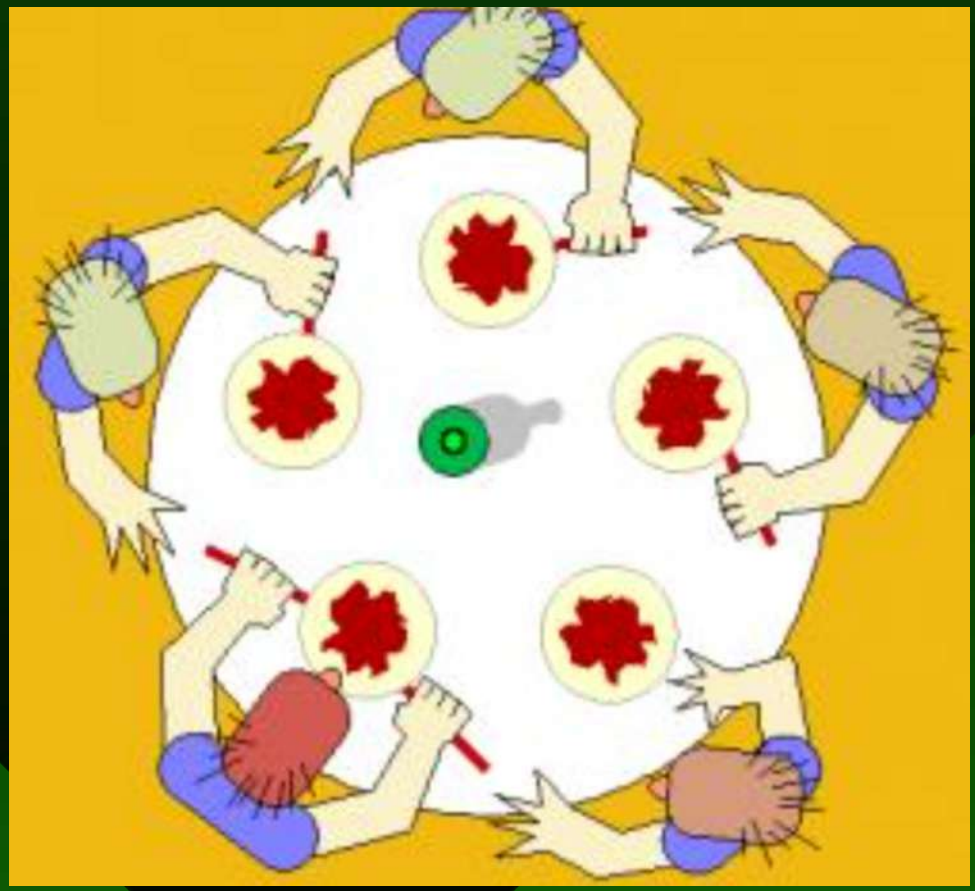
Como garantir que o consumidor não vai remover dados de um *buffer* vazio?

A solução não é trivial, pois os processos podem ser interrompidos! O que aconteceria se, antes de concluir o incremento de um contador, o processo fosse interrompido, e o contador fosse decrementado pelo outro processo?

Sincronização de Processos

O jantar dos filósofos

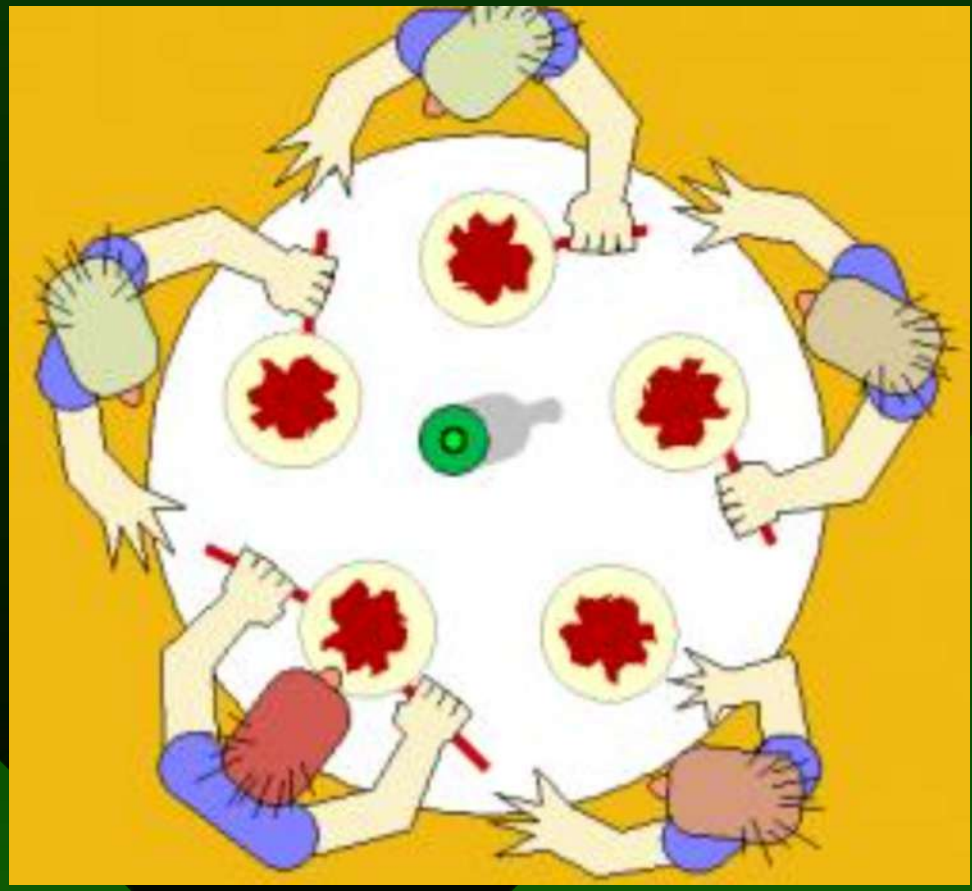
- Há cinco filósofos em torno de uma mesa;
- Um garfo é colocado entre cada filósofo;
- Cada filósofo deve, alternadamente, refletir e comer;
- Para que um filósofo coma, ele deve possuir dois garfos



Sincronização de Processos

O jantar dos filósofos

- O garfo somente pode ser pego se não estiver em uso por nenhum outro filósofo;
- Após comer, o filósofo deve liberar o garfo que utilizou;
- Um filósofo pode segurar o garfo da sua direita ou o da sua esquerda assim que estiverem disponíveis, mas só pode começar a comer quando ambos estiverem sob sua posse.



Aula 06

Sincronismo de Processos (continuação)

Resolvendo o sincronismo

Existem soluções de *hardware* e *software*;

Definição da "Região Crítica"

- Exclusão Mútua controlada por variável?
- Desabilitar interrupções?

Processos / *threads* que aguardam

- "Espera Ocupada";
- Bloqueio?

Região Crítica e Exclusão Mútua

Como tratar o acesso a recursos globais compartilhados, com instruções intercaladas pelo escalonador, fora do controle do programador?

Acesso realizado de forma atômica

Modelo de Código:

```
{ ***
SeçãoNãoCrítica;
ProtocoloDeEntrada;
SeçãoCrítica;
ProtocoloDeSaída;
****
}
```

Região Crítica e Exclusão Mútua

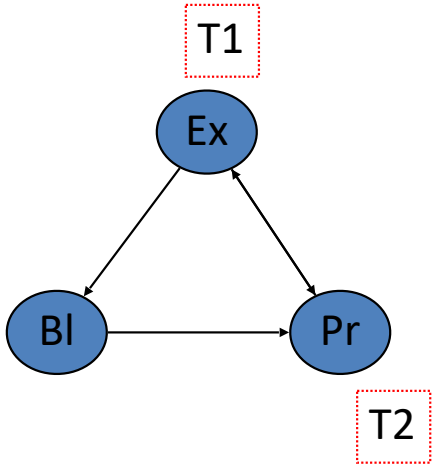
A região crítica é um trecho do código que deve ser executado de forma atômica, sem a intercalação com outro processo;

Metas:

- APENAS UM processo/*thread* estará na sua sessão crítica de cada vez, ou seja, deverá ser executado sob exclusão mútua;
- Um processo/*thread* não pode ser interrompido ou entrar em loop dentro dos protocolos de entrada/saída ou na sessão crítica;
- Nenhum processo/*thread* fora da sessão crítica pode bloquear outro processo/*thread*;

Threads Concorrentes

Chave = 1



- Thread 1

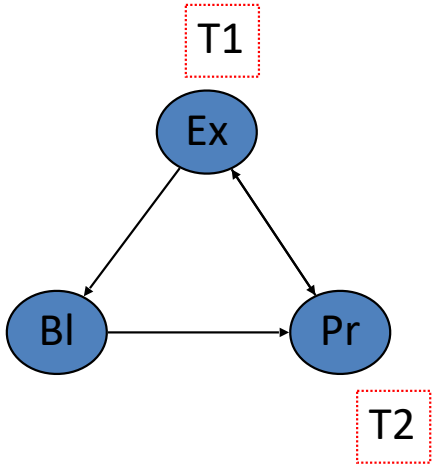
```
...
while (chave = = 0);
chave = 0;
RC
chave = 1;
...
```

- Thread 2

```
...
while (chave = = 0);
chave = 0;
RC
chave = 1;
...
```

Threads Concorrentes

Chave = 0



- Thread 1

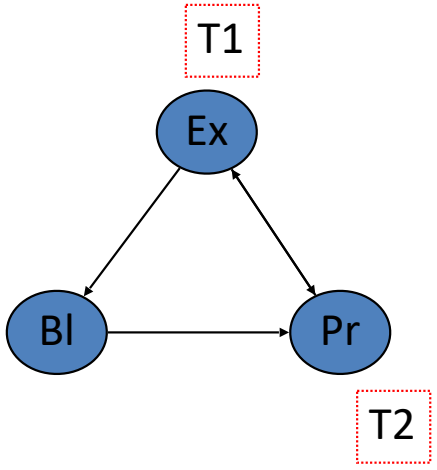
```
...
    while (chave = = 0);
    chave = 0;
     RC
    chave = 1;
...
```

- Thread 2

```
...
    while (chave = = 0);
    chave = 0;
     RC
    chave = 1;
...
```

Threads Concorrentes

Chave = 0



- Thread 1

```
...
   while (chave = = 0);
  chave = 0;
   RC
  chave = 1;
...
```

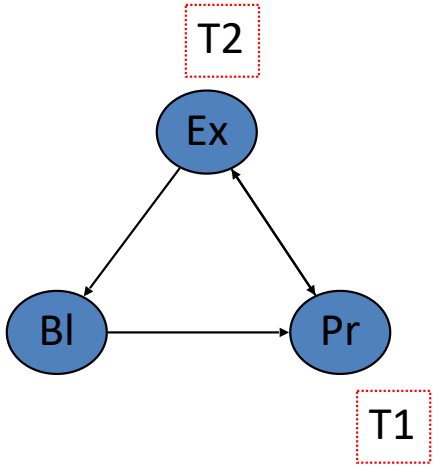
Preempção
Por tempo

- Thread 2

```
...
   while (chave = = 0);
  chave = 0;
   RC
  chave = 1;
...
```

Threads Concorrentes

Chave = 0



- Thread 1

```
...
   while (chave = = 0);
  chave = 0;
   RC
  chave = 1;
...
```

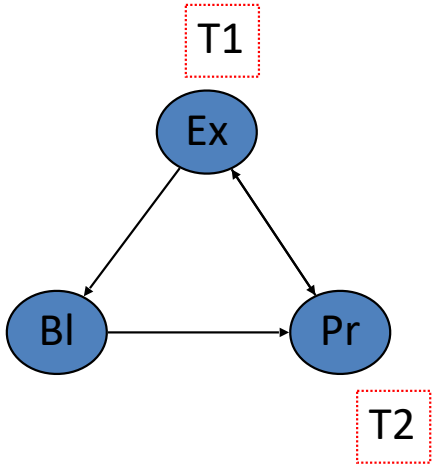
- Thread 2

```
...
   while (chave = = 0);
  chave = 0;
   RC
  chave = 1;
...
```

A thread 2 vai passar todo o tempo de quantum testando se chave = 0.

Threads Concorrentes

Chave = 0



- Thread 1

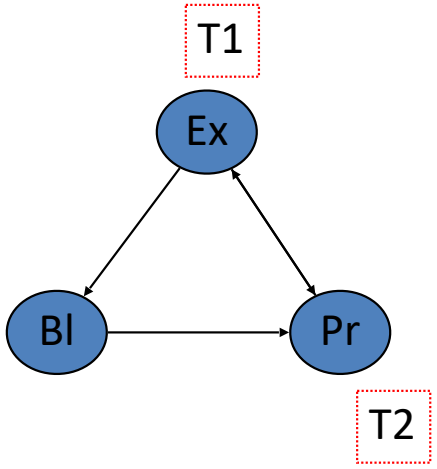
```
...
   while (chave = = 0);
  chave = 0;
   RC
  chave = 1;
...
```

- Thread 2

```
...
   while (chave = = 0);
  chave = 0;
   RC
  chave = 1;
...
```

Threads Concorrentes

Chave = 1



- Thread 1

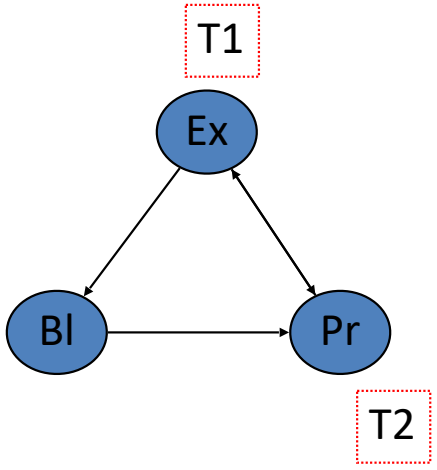
```
...
    while (chave = = 0);
   chave = 0;
    RC
→ chave = 1;
...
```

◆ Thread 2

```
...
→  while (chave = = 0);
   chave = 0;
    RC
   chave = 1;
...
```

Threads Concorrentes

Chave = 1



- Thread 1

```
...
   while (chave = = 0);
  chave = 0;
    RC
  chave = 1;
...
```

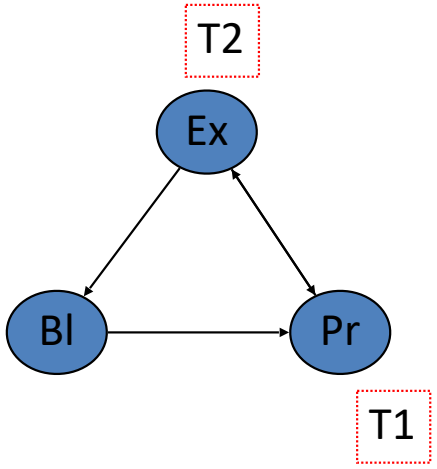
Preempção
Por tempo

◆ Thread 2

```
...
   while (chave = = 0);
  chave = 0;
    RC
  chave = 1;
...
```

Threads Concorrentes

Chave = 1



- Thread 1

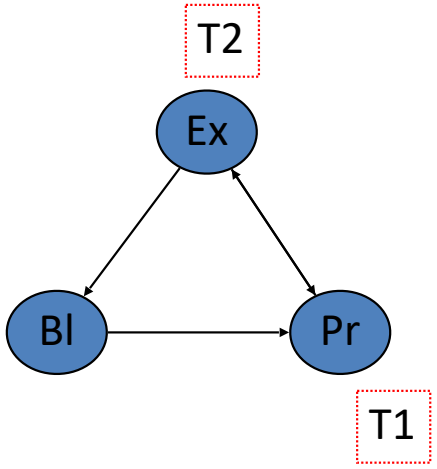
```
...
    while (chave = = 0);
   chave = 0;
     RC
   chave = 1;
...
```

- Thread 2

```
...
    while (chave = = 0);
   chave = 0;
    RC
   chave = 1;
...
```

Threads Concorrentes

Chave = 0



- Thread 1

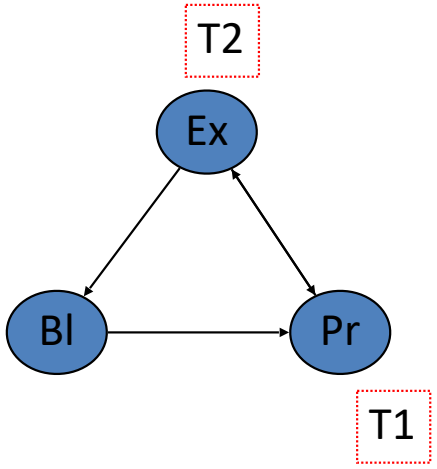
```
...
    while (chave = = 0);
   chave = 0;
    RC
   chave = 1;
...
```

◆ Thread 2

```
...
    while (chave = = 0);
   chave = 0;
    RC
   chave = 1;
...
```

Threads Concorrentes

Chave = 0



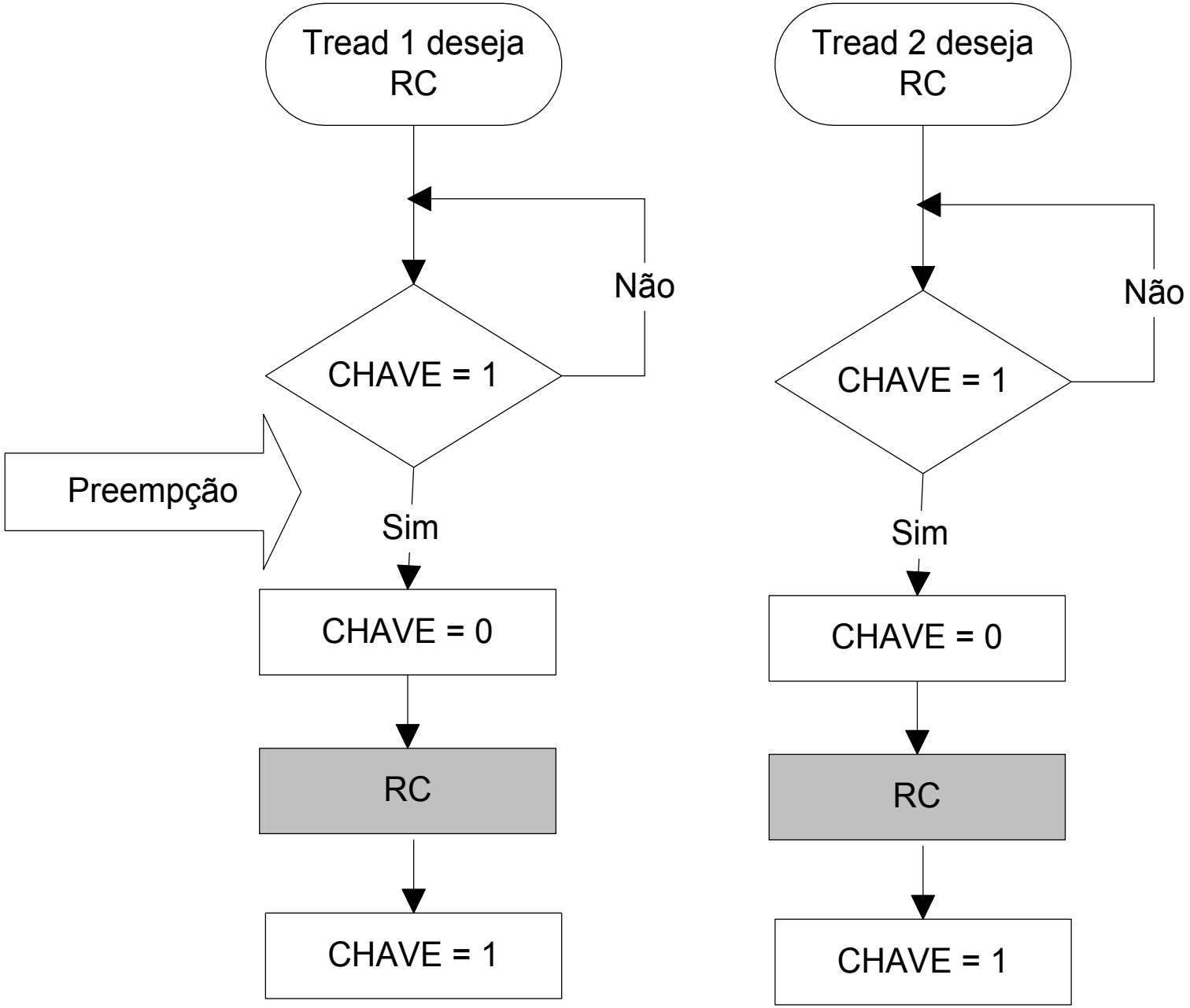
- Thread 1

```
...
    while (chave = = 0);
   chave = 0;
     RC
   chave = 1;
...
```

◆ Thread 2

```
...
    while (chave = = 0);
   chave = 0;
    RC
   chave = 1;
...
```

Existem problemas? Sim !



O ponto de falha é o teste !

Entre o teste e a atribuição de valor para o semáforo pode haver preempção?

- Sim ! A interrupção aguarda apenas o final da instrução corrente.

Soluções de *hardware*:

- Desabilitar interrupções;
- Usar uma nova instrução.

O ponto de falha é o teste !

Desabilitar Interrupções

- Pode comprometer o desempenho e integridade do sistema se não for reabilitada;
- Difícil em ambiente multiprocessado
  - Desabilita interrupção em 1 processador, e ... os outros continuam a executar processos/*threads*
- *Clock* é atualizado através de interrupções;
- Sem interrupções não há troca de processos por estouro de tempo.

O ponto de falha é o teste !

- Instrução *Test-And-Set* (TSL)
  - Uma das instruções do processador
  - Em uma única instrução:
    - Lê variável
    - Armazena valor num registrador
    - Atribui novo valor à variável
  - Execução atômica
    - Não pode ser interrompida

Sleep-WakeUp

A solução vista até agora envolve a chamada "espera ocupada", ou seja, perde-se tempo valioso do processador em um "*loop*" até o estouro do *Quantum*;

O uso do recurso de "*Sleep-WakeUp*" poderia bloquear o processo que encontra a região crítica "fechada".

Condições para o *Deadlock*

A preocupação com a execução simultânea de processos fez, na prática, que um processo aguardasse o término da execução de outro;

Mas o que acontece quando dois ou mais processos aguardam um pelos outros?

Condições para o *Deadlock*

Condições necessárias:

1. Exclusão Mútua
   Apenas um processo por vez pode alocar e manipular um recurso
2. Posse e Espera;
3. Não Preempção;
4. Espera Circular.

Ocorrência precisa ser simultânea

Se ao menos uma não ocorrer, não haverá *DeadLock*

Condições para o *Deadlock*

Condições necessárias:

1. Exclusão Mútua
2. Posse e Espera

   Um processo, de posse de um recurso, pode solicitar novos recursos
3. Não Preempção;
4. Espera Circular.

Ocorrência precisa ser simultânea

Se ao menos uma não ocorrer, não haverá *DeadLock*

Condições para o *Deadlock*

Condições necessárias:

1. Exclusão Mútua
2. Posse e Espera
3. Não Preempção

   Um recurso não pode ser removido explicitamente do processo
4. Espera Circular.

Ocorrência precisa ser simultânea

Se ao menos uma não ocorrer, não haverá *DeadLock*

Condições para o *Deadlock*

Condições necessárias:

1. Exclusão Mútua
2. Posse e Espera
3. Não Preempção
   Um recurso não pode ser removido explicitamente do processo
4. Espera Circular
   Ocorre CICLO no GRAFO de alocação

Ocorrência precisa ser simultânea

Se ao menos uma não ocorrer, não haverá *DeadLock*